The Design and Analysis of Real-World Cryptographic Protocols

Samuel Scott

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
Department of Mathematics
Royal Holloway, University of London

2017

Declaration

These doctoral studies were conducted under the supervision of Prof. Kenneth G. Paterson and Prof. Simon R. Blackburn.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Samuel Scott
October, 2017

Acknowledgements

First of all I would like to sincerely thank my supervisor Kenny Paterson. Kenny is the most dedicated researcher I have met, and despite being incredibly busy would always take the time to help me when it mattered most.

Thanks to Kenny, I had the great fortune to work with many other incredible people, including all of my wonderful co-authors. In particular, I want to thank Tom Ristenpart, who has been a pleasure to work with, and for inviting me to work on Pythia, which led to working with the amazing Ari Juels, Rahul Chatterjee, and Adam Everspaugh. Similarly, thanks to Eric Rescorla for taking me on as an intern at Mozilla. Not only was Mozilla a fantastic experience, but it led to the rewarding collaboration with Cas Cremers, Marko Horvat, Thyla van der Merwe, and Jonathan Hoyland, all of whom I have shared many laughs with over the years. I also want to thank Simon Blackburn for supervising me in my first year, who was a fantastic supervisor and a pleasure to work with, and I am incredibly grateful to have had the chance to work with Martin Albrecht and Rachel Player.

Thank you to the EPSRC and Royal Holloway for conceiving of and funding this PhD programme.

Of course, none of this would have been possible without the support and encouragement of my wife Yiran. Thank you for pushing me to undertake a PhD when I first had doubts, for keeping me sane when deadlines loomed, and cooking amazing food when I was stressed.

I want to thank everyone at Royal Holloway for all the good times we spent together, and many unforgettable memories. Special mention to Thalia and Dan, who I've known since the MSc programme, Thalia for all the tea and cake, and Dan who was responsible for me first enquiring about doing a PhD.

Finally I thank all my friends and family, wonderful people to whom I owe a lot, especially thanks to my Dad who has been a role model for me in everything I do.

Abstract

Designing cryptographic protocols for use in the real world is a challenging task, requiring a fine balance between practicality and security. Ad hoc constructions are often catastrophically broken, and even well-studied protocols regularly do not stand up to the test of time. We look at some of the ways cryptographic protocols are designed and analysed, applying these techniques to a variety of real-word scenarios.

Our first scenario considers password storage, introducing a new primitive called a verifiable, partially oblivious PRF. We analyse the suitability of this primitive to the application in question, provide formal security proofs, and evaluate an example instantiation.

The second scenario introduces a new security model to better understand the domain of key rotation for authenticated encryption. This is an area highly relevant to modern practices of storing data encrypted in the cloud. By introducing this new security model, we show that existing solutions fall short of achieving any meaningful security properties, and suggest some simple fixes. Finally, we implement and prove a new construction which meets our strongest definition, and analyse its practicality.

Finally, to contrast with the computational approach in previous chapters, we additionally consider symbolic approaches to security analysis, using the formal verification tool TAMARIN to prove security properties of the latest draft of the TLS 1.3 specification. Our results show formal method complement other approaches nicely, and provide a new perspective.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an overview of the thesis. We provide the motivation for our research and describe the contributions of this thesis. In this chapter, we also present the overall structure of the thesis.

1.1 Motivation

This thesis is motivated by taking the exciting research produced in recent years by the cryptographic community, and applying it to real-world applications. However, the journey from novel cryptographic invention to a fully-specified, production-ready implementation is a long one. Great care must be taken at each step along the path to prove the resultant scheme meets the desired properties.

Furthermore, not all paths start from the same place. In the design of Pythia, a modern PRF service, we start with a novel cryptographic primitive, resulting in a simple two-party protocol and investigate applications in which this protocol turns out to be of great use.

On the other hand, sometimes the starting point is the application itself. Of particular importance is when cryptography is already being used to fulfil a certain requirement with neither a careful study of the assumed security properties nor their implications.

We identified key rotation for authenticated encryption as one such area. The current methods to achieve rotation are simple, ad-hoc constructions with no formal backing. We rectify this by taking a thorough theoretical approach to security, and propose a model in which to properly analyse these constructions. This approach identified limitations of the existing approaches used in practice, and we suggested some simple tweaks to achieve higher levels of security.

Finally, in contrast with the previous examples which are novel and previously unexplored, we studied TLS, one of the most widely used cryptographic protocols. More specifically, we analysed the proposed specification for the latest version, TLS 1.3. Our methodology focused on the use of symbolic methods to analyse protocol specifications, highlighting a number of important considerations and ultimately influencing the final specification.

We expand on each of these examples in what follows.

1.1.1 Password Storage

Conventional cryptographic services such as hardware-security modules (HSMs) and software-based key-management systems offer the ability to apply a pseudorandom function (PRF) such as HMAC to inputs of a client's choosing. These services are used, for example, to harden stored password hashes against offline brute-force attacks, and are used by Facebook for this purpose [149].

We propose a modern PRF service called Pythia designed to provide all of the benefits of a local pseudorandom function, while offering a significantly more flexible, and practical deployment scenario.

The keystone of Pythia is a new cryptographic primitive called a *verifiable partially-oblivious PRF* that reveals a portion of an input message to the service but hides the rest. We give a construction that additionally supports efficient bulk rotation of previously obtained PRF values to values under new keys. Performance measurements show that our construction, which relies on bilinear pairings and zero-knowledge proofs, is highly practical. We also give accompanying formal definitions and proofs of security.

We implement Pythia as a multi-tenant, scalable PRF service that can scale up to hundreds of millions of distinct client applications on commodity systems. In our prototype implementation, query latencies are 15 ms in local-area settings and throughput is within a factor of two of a standard HTTPS server. We further report on implementations of two applications using Pythia, showing how to bring its security benefits to a new enterprise password storage system and a new brainwallet system for Bitcoin.

1.1.2 Key Rotation for Authenticated Encryption

A common requirement in practice is to periodically rotate the keys used to encrypt stored data. Systems used by Amazon and Google do so using a hybrid encryption technique which is eminently practical but has questionable security in the face of key compromises and does not provide "full" key rotation. Meanwhile, symmetric updatable encryption schemes (introduced by Boneh *et al.* [60]) support full key rotation without performing decryption: ciphertexts created under one key can be rotated to ciphertexts created under a different key with the help of a re-encryption token. By design, the tokens do not leak information about keys or plaintexts and so can be given to storage providers without compromising security. But the prior work of Boneh *et al.* addresses relatively weak confidentiality goals and does not consider integrity at all. Moreover, as we show, a subtle issue with their concrete scheme obviates a security proof even for confidentiality against passive attacks.

We present a systematic study of what we call *updatable Authenticated Encryption*. We provide a set of security notions that strengthen those in prior work. These notions enable us to tease out real-world security requirements of different strengths and build schemes that satisfy them efficiently. We show that the hybrid approach currently used in industry achieves relatively weak forms of confidentiality and integrity, but can be modified at low cost to meet our stronger confidentiality and integrity goals. This leads to a practical scheme that has negligible overhead beyond conventional authenticated encryption (AE). We then introduce *re-encryption indistinguishability*, a security notion that formally captures the idea of fully refreshing keys upon rotation. We show how to repair the scheme of Boneh *et al.*, attaining our stronger confidentiality notion. We also show how to extend the scheme to provide integrity, and we prove that it meets our re-encryption indistinguishability notion. Finally, we discuss how to instantiate our scheme efficiently using off-the-shelf cryptographic components (authenticated encryption, hashing, elliptic curves). We report on the performance of a prototype implementation, showing that fully secure key rotations can be performed at a throughput of approximately 139 kB/s.

Although our work on updatable AE may give the impression of a natural process to define a security model for key rotation, in reality the model was the result of a lengthy process, iterating between strong security definitions, and practical concerns. In the process, we ruled out many constructions which were flawed in subtle ways, and established a common set of techniques for achieving security. It is certain that without this rigorous process to define security as tightly and securely as possible, we would not have arrived at the final construction.

1.1.3 Transport Layer Security (TLS)

The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new version of the protocol, namely TLS 1.3. Over the past three years, in an unprecedented joint design effort with the academic community, the TLS Working Group of the IETF has been working steadily to enhance the security of TLS.

We contribute to this effort by constructing the most comprehensive, faithful, and modular symbolic model of the TLS 1.3 release candidate to date, and use the Tamarin prover to verify the claimed TLS 1.3 security requirements, as laid out in revision 21 of the specification. In particular, our model covers *all* handshake modes of TLS 1.3.

Our analysis reveals a previously unreported unexpected behaviour, which we expect will inhibit strong authentication guarantees in some implementations of the protocol. In contrast to previous models, we provide a novel way of making the relation between the TLS specification and our model explicit: we provide a fully annotated version of the specification that clarifies what protocol elements we modelled, and precisely how we modelled these elements. We anticipate this model artefact to be of great benefit to the academic community and the TLS Working Group alike.

We also provide details of our prior work to model revision 10 of the specification. At the time, we further extended our model to incorporate the desired delayed client authentication mechanism, and uncovered a potential attack in which an adversary is able to successfully impersonate a client during a PSK-resumption handshake.

In the context of this thesis, this work showcases symbolic modelling as an alternate approach to the analysis of protocols. Our model of TLS 1.3 is highly sophisticated, modelling many different components and, most importantly, the interaction between these components in a single framework. This comprehensive approach aims to unearth any awkward flaws which arise from incompatible assumptions across diverse parts of the protocol. For example, the aforementioned attack resulted from a sequence of events including unilateral authentication, session resumption, and bilateral authentication.

1.2 Thesis Structure

Chapter 2 includes a brief background on some important concepts and definitions in the areas of both computational and symbolic analysis. These are not designed to be comprehensive introductions, but rather serve as a solid foundation for the rest of the thesis. In particular, this chapter will introduce many of the customs and conventions used in the rest of the thesis.

Chapter 3 discusses the Pythia PRF service. We start with the creation of a new cryptographic primitive – a verifiable, partially-oblivious PRF – which serves as the core primitive of Pythia, and define a new model to capture the desired security properties that it should possess. We detail a concrete instantiation of Pythia which satisfies these security notions and is practically applicable. Finally, we analyse the applicability of Pythia to two real-world use cases: password hardening and brain-wallets.

Chapter 4 covers the area of key rotation for authenticated encryption. We start with an exposition of the problem itself, and the unsuitability of current approaches to provide meaningful security guarantees. We define a new set of security properties and provide constructions achieving these goals. Finally, we analyse our implementations of these constructions.

In contrast with the previous chapters, in Chapter 5 we apply symbolic analysis techniques. The object of our analysis is the TLS 1.3 protocol specification. and we present our early results on revision 10 as well as the latest work on revision 21 – suggested to be close to the final specification.

1.3 Associated Publications

In the course of my PhD, I have been fortunate to work on some diverse projects, and with many talented collaborators. In the following, I will expand on my role in each of these pieces of work, but wish to emphasise that all work was a fully collaborative effort, and could not have been possible without the contributions of all co-authors.

- Adam Everspaugh, Rahul Chatterjee, Samuel Scott, Ari Juels, Thomas Ristenpart, and Cornell Tech. "The Pythia PRF service". In: *Proceedings of the 24th USENIX Conference on Security Symposium.* USENIX Association. 2015, pp. 547–562.

  I became involved on this project after discussing the application of key- homomorphic PRFs to password hardening with Thomas Ristenpart. Coincidentally, his team had been working on precisely this area (a PRF-based password hardening service), and the addition of key rotation offered great benefits. Therefore, I was invited to join the work, and was tasked with proving the security of our scheme, as well as helping to expand on the applications.

- Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. "Key Rotation for Authenticated Encryption". In: *Advances in Cryptology – CRYPTO 2017, Part III.* Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2017, pp. 98–129.

  This work was the result of a lengthy project to determine a suitable formulation of updatable encryption. The main theoretical aspects of the work were due to frequent discussions between Kenny Paterson, Thomas Ristenpart, and I. Adam Everspaugh was later invited to join the work, primarily to help with the practical implementations together with myself, but was also involved in finalising some of the theoretical aspects.

- Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". In: *2016 IEEE Symposium on Security and Privacy.* San Jose, CA, USA: IEEE Computer Society Press, May 2016, pp. 470–485

Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. “A Comprehensive Symbolic Analysis of TLS 1.3”. In: *Proceedings of ACM CCS 2017: 24th Conference on Computer and Communications Security*. 2017.

These two papers were the result of a long and extremely fruitful project, initially started when Thyla van der Merwe and I were employed at Mozilla as interns and tasked by Eric Rescorla to help analyse the security of the TLS 1.3 specification.

A large portion of the modelling effort was undertaken by Thyla and I while at Mozilla, which I continued after the conclusion of our 3-month internships. We were supported by Cas Cremers and Marko Horvat in initially learning the Tamarin tool, and with frequent discussions on aspects of the theory and modelling.

The major rewrite of the model for our second paper was largely performed by myself, supported by my co-authors, and we additionally invited Jonathan Hoyland to help with this effort. As part of this rewrite, I also produced the side-by-side documentation, which has been well-received as a transparency tool.

Additionally, the following work was conducted in the course of this PhD, but is not included in this thesis:

- Simon R. Blackburn and Sam Scott. “The discrete logarithm problem for exponents of bounded height”. In: *LMS Journal of Computation and Mathematics* 17 (Special Issue A Jan. 2014), pp. 148–156. ISSN: 1461-1570.
- Martin R Albrecht, Rachel Player, and Sam Scott. “On the concrete hardness of learning with errors”. In: *Journal of Mathematical Cryptology* 9.3 (2015), pp. 169–203.

CHAPTER 2

Background

Contents

In this chapter, we introduce some of the basic definitions, notation and concepts used throughout the thesis.

2.1 Notation

We express constructions, protocols, and security games in pseudocode. The games themselves are written as procedures, which can be thought of as being run by the challenger. The adversary $\mathcal{A}$ is typically given a set of oracles, written $\mathcal{A}^{(O_1,O_2,\dots)}$. In keeping with the pseudocode approach, we sometimes will refer to array lookups of the form $A[i]$, and use object notation $P.foo$ to represent the variable or procedure foo owned by P.

In the procedures, we write $x \leftarrow 0$ for variable assignment and $x \leftarrow_{\$} \mathcal{S}$ for sampling x from some set $\mathcal{S}$ (uniformly unless a distribution is specified). The same notation $y \leftarrow f(x)$ and $y \leftarrow_{\$} f(x)$ are respectively used to represent that y is returned by the function $f(x)$ deterministically, or from the randomised function $f(x)$ with some implicit source of randomness.

We write $x = y$ for the logical check "is x is equal to y".

The symbol $\perp$ is used to represent a generic failure within pseudocode.

2.2 Computational Analysis

In this section we provide examples of the computational approach to cryptographic analysis, also commonly referred to as provable security.

This is a broad topic, and here we primarily provide examples which are relevant for later chapters.

Many cryptographic definitions and security models fit within one of a few common paradigms. The benefit of re-using common paradigms is similar to that of re-using common notation: people already familiar with the field are able to quickly grasp new concepts.

We give a few simple definitions here as an example of the typical syntax and structure of definitions used throughout this thesis. For a more detailed introduction, we emphasise that a reader should consult an introductory text such as Katz and Lindell [119].

2.2.1 Concrete Security

In this thesis, we try to follow the approach of Bellare and Rogaway to provable security, that of *practice-oriented provable-security* as discussed by Bellare in an invited talk and accompanying article [27], and by Rogaway in [169].

One aspect of this is concrete security, which aims to provably quantify the security of a scheme. This is typically achieved through security reductions, which say that the strength of a scheme is related to the strength of its components; the reduction may additionally be constructive, in which case breaking the target scheme produces a concrete adversary attacking an underlying primitive. A distinguishing feature of this approach is that the theorem statement gives a lower bound on the amount of time an attacker would take to break a particular instantiation of a scheme.

The alternative approach is considering asymptotic security, in which schemes should be secure if a polynomial-time adversary cannot break it. Theorem statements are quantified over families of schemes, with some security parameter. The implication being that one can simply increase the security parameter to increase the difficulty of breaking scheme. The concrete approach is more suitable for practice-oriented work, and hence this thesis.

In this thesis, we strongly favour game-based definitions. In such definitions, a certain security property is encapsulated by setting up an experiment in which an adversary must fulfil a certain condition in order to "win".

For example, the classical notion of "left-or-right" encryption, also known as IND-CPA (indistinguishability of encryption against chosen-plaintext attacks), which we define in the following section, can be formulated by defining a game in which the adversary needs to guess a hidden bit b. Here the "win" condition is whether or not the adversary guesses b correctly.

We define the *advantage* to be the probability that an adversary wins in a particular security game. The advantage is parametrised by the specific game, the target scheme, and frequently, the resources of the attack. The resources includes quantities such as the number of queries made, the computation time, or the total length of messages sent.

We informally use the term *security level* of a scheme to refer to the upper bound over all advantage terms, parametrised by the required resources.

A security reduction is a relation between two advantage terms. We commonly use reductions to produce bounds on a class of adversaries in relation to another adversary. In particular, a reduction can be used to show that the security level of a certain scheme is purely dependent on the security of a certain primitive.

Two security games are said to be equivalent if there exists a security reduction showing equality between adversaries of the two games for any given scheme.

2.2.2 Symmetric Encryption

We do not cover all the details of symmetric encryption, nor intend this to be a comprehensive introduction, but present these definitions as a guide to the definitional syntax used throughout. A thorough introduction to symmetric encryption can be found in the lecture notes of Bellare [26].

We start with a basic definition of a symmetric encryption scheme.

Definition 1 (Symmetric encryption scheme)**.** A symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a tuple of algorithms with the following properties:

- $\mathcal{K}() \to k$. Outputs a secret key k. In many cases k is sampled uniformly at random from a keyspace, and we write $k \leftarrow_{\$} \mathcal{K}$ for convenience.
- $\mathcal{E}(k, m) \to C$. On input a secret key k and message $m \in \{0,1\}^*$, outputs a ciphertext $C \in \{0,1\}^* \cup \{\perp\}$. We sometimes write $C \leftarrow_{\$} \mathcal{E}_k(m)$ for convenience.
- $\mathsf{Dec}(k, C) \to m$. On input a secret key k and ciphertext $C \in \{0,1\}^*$ outputs a message $m \in \{0,1\}^* \cup \{\perp\}$. We sometimes write $m \leftarrow_{\$} \mathcal{D}_k(C)$ for convenience.

Recall that $\perp$ is the generic error symbol, representing either an encryption or decryption failure, in which case no other information is returned.

A symmetric encryption scheme should have the correctness property, which means that for all keys $k \leftarrow_{\$} \mathcal{K}$, all messages m, and any ciphertext $C \leftarrow_{\$} \mathcal{E}_k(m)$, it should be the case that $\mathcal{D}_k(C) = m$ whenever $C \neq \perp$.

Confidentiality

The IND-CPA security game, which we introduce in the following, is one of the fundamental security definitions in cryptography. Although having a simple formulation, it captures a strong definition of confidentiality for symmetric encryption. The general idea is that the adversary provides two messages (of the same length) for encryption, one of which is subsequently chosen and encrypted by an oracle. The adversary is tasked with determining which message was encrypted.

On first appearance, it is not clear that this is a “good” definition of confidentiality. It does not appear to say anything about the capability of the adversary to learn anything about the plaintext, given an encrypted message. This property is more directly captured by a notion called semantic security. However, it can be shown that the IND-CPA game is equivalent to semantic security, the intuition being that if the adversary can learn anything about which message was encrypted, then they can use this advantage to distinguish ciphertexts in the IND-CPA game. For more details on semantic security and a formal proof of the relation, we refer to [26].

This scenario hints towards the power of provable security. We have a strong definition of security – semantic security. On the other hand, we have another definition with a simpler formulation. By showing their equivalence, we can use the simple formulation, safe in the knowledge that proving one implies the other.

In the following, we give a formal definition of the IND-CPA game, and corresponding notion of security.

Definition 2 (IND-CPA Game)**.** Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. In the IND-CPA game, a bit b is sampled from $\{0, 1\}$, and a random key $k \leftarrow\!\!\$\, \mathcal{K}$ is generated. The adversary $\mathcal{A}$ is given access to an encryption oracle Enc, which returns on input (m_0, m_1) the encryption of the message m_b under key k. The game outputs `true` if the adversary at the end of the game correctly returns the bit $b' = b$.

The game is more formally described in Figure 2.1

The advantage $\mathsf{Adv}_{\Pi}^{\text{ind-cpa}}$ for an adversary $\mathcal{A}$ in the IND-CPA game attacking Π is the probability that the adversary correctly guesses the bit b:

$$\mathsf{Adv}_{\Pi}^{\text{ind-cpa}}(\mathcal{A}) = 2 \cdot \Pr[\text{IND-CPA} \Rightarrow \mathsf{true}] - 1.$$

In the definition of the IND-CPA game we assume the adversary only queries Enc for messages (m_0, m_1) of (pairwise) equal length.

Note that the scaling applied to the success probability is a common trick to normalise the advantage to a value between 0 and 1, representing respectively that the adversary has either no chance of winning better than guessing, or that they correctly return the bit b with certainty.

For a scheme to be *secure* in the sense of IND-CPA, we require that for all adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}_{\Pi}^{\text{ind-cpa}}(\mathcal{A})$ is sufficiently small as to be practically infeasible. The advantage of the adversary is often proportional to the number of queries made, or the length of those queries (whether measured in bits or blocks). In this case,

```
IND-CPA
$k \leftarrow\!\!\$\, \mathcal{K}$
$b \leftarrow\!\!\$\, \{0,1\}$
$b' \leftarrow\!\!\$\, \mathcal{A}^{\mathrm{Enc}}()$
return $(b' = b)$

Enc($m_0, m_1$)
return $\mathcal{E}(k, m_b)$
```

```
INT-CTXT
$k \leftarrow\!\!\$\, \mathcal{K}$
win $\leftarrow$ false
$S \leftarrow \emptyset$
$\mathcal{A}^{(\mathrm{Enc},\mathrm{Dec})}()$
return win

Enc($m$)
$C \leftarrow\!\!\$\, \mathcal{E}(k, m)$
$S \leftarrow S \cup C$
return $C$

Dec($C$)
$m \leftarrow \mathcal{D}(k, C)$
if $m \neq \perp \wedge C \notin S$ then
    win $\leftarrow$ true
return $m$
```

Figure 2.1: Left: The IND-CPA game. Right: The INT-CTXT game.

we may also parametrise the advantage over these terms. For example, we define $\mathsf{Adv}^{\text{ind-cpa}}_{\Pi,q}(\mathcal{A})$ to be the advantage of an adversary $\mathcal{A}$ making at most q queries to the Enc oracle.

Suppose, for example, we only consider adversaries $\mathcal{A}$ making at most q queries, where each query is precisely one block, and suppose we show that for all such adversaries $\mathsf{Adv}^{\text{ind-cpa}}_{\Pi,q}(\mathcal{A}) \leq \frac{q^2}{2^n}$, where n is the key-length. If n is 64, then an adversary would need to make at least 2^{32} queries to achieve distinguishing advantage 1. In many scenarios, this would be insufficiently secure. For a simple encryption scheme, this might represent $2^{32} \times 64\text{b} = 32\text{GiB}$ of encryptions. This is certainly within the realms of a realistic scenario.

On the other hand, if the advantage term is $\frac{q}{2^n}$ and n is 128, then even 2^{64} queries results in a distinguishing advantage of just $\frac{1}{2^{64}}$. This is almost certainly secure for most realistic scenarios.

Hence we do not consider IND-CPA security to be a fixed, absolute notion, but rather a quantitative term. Instead, we produce results which measure the security a scheme achieves under different threat models, which helps to inform protocol designers about precisely how secure a scheme is.

Integrity

Another important definition for symmetric encryption schemes is the notion of integrity. For example, INT-CTXT (integrity of ciphertexts) requires that the adversary produce a *valid* ciphertext distinct from any previously seen, when given access to encryption and decryption oracles.

Definition 3 (INT-CTXT Game)**.** Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. In the INT-CTXT game, an adversary $\mathcal{A}$ is given access to oracles Enc, Dec, to which it can make encryption and decryption queries respectively.

The INT-CTXT game outputs `true` if the adversary at some point during the game queries the Dec oracle with a ciphertext C, such that C was not previously output by a query to Enc, and C is a valid ciphertext, i.e., $\mathcal{D}(k, C) \neq \perp$.

The game and associated oracles are depicted in Figure 2.1.

The advantage $\mathsf{Adv}_{\Pi}^{\text{int-ctxt}}$ for an adversary $\mathcal{A}$ in the INT-CTXT game attacking Π is the probability that the INT-CTXT game outputs true

$$\mathsf{Adv}_{\Pi}^{\text{int-ctxt}}(\mathcal{A}) = \Pr\left[\text{INT-CTXT} \Rightarrow \mathsf{true}\right].$$

A scheme Π is secure in the sense of INT-CTXT if for any adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\Pi}^{\text{int-ctxt}}(\mathcal{A})$ is "small". As before, we define $\mathsf{Adv}_{\Pi,q}^{\text{int-ctxt}}(\mathcal{A})$ to be the advantage of an adversary making at most q queries to the Enc and Dec oracles.

Authenticated Encryption

Finally, to complete this short section on security definitions for symmetric encryption, we combine the previous two into a single all-in-one definition - the authenticated encryption game due to [171, 167]:

Definition 4 (Authenticated Encryption Game)**.** Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme.

We define the oracles used in the AE-ROR game as follows:

- Enc: takes as input a bit string m and produces $\mathcal{E}_k(m)$ when $b = 0$, and produces a random string of the same length otherwise.
- Dec: takes as input a bit string C and produces $\mathcal{D}_k(C)$ when $b = 0$, and returns $\perp$ otherwise.

In the AE-ROR game, the hidden values $b \in \{0, 1\}$ and $k \leftarrow_{\$} \mathcal{K}$ are sampled uniformly at random at the start of the game. The adversary $\mathcal{A}$ is given access to the Enc and Dec oracles and must output a bit b'. The game outputs true when $b = b'$. We further require that the adversary does not submit outputs from the Enc oracle to the Dec oracle.

We define the advantage of $\mathcal{A}$ in the AE-ROR game for π to be:

$$\mathsf{Adv}_{\Pi}^{\text{ae}}(\mathcal{A}) = 2 \cdot \Pr\left[\text{AE-ROR}_{\pi}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1.$$

Then the security of Π in the AE-ROR is the upper bound of $\mathsf{Adv}^{\mathrm{ae}}_{\Pi}(\mathcal{A})$ over all adversaries $\mathcal{A}$.

As usual, when quantifying the advantage over, for example, number of queries, we define $\mathsf{Adv}^{\mathrm{ae}}_{\Pi,q}(\mathcal{A})$ to be the advantage of an adversary $\mathcal{A}$ making at most q queries to either the Enc or Dec oracles.

In the above AE-ROR game, there are two small modifications from the IND-CPA and INT-CTXT games. First, instead of the bit b determining whether the encryption is m_0 or m_1 (left-or-right), we have that b determines whether a true encryption $\mathcal{E}(k, m)$ is returned or a randomly generated string (real-or-random). It can be shown that the real-or-random formulation (called IND\$-CPA) is equivalent to IND-CPA, in that the security is within a constant factor of two [28], when Π is a length-regular scheme.

Additionally, instead of requiring that the adversary needs to submit a valid, distinct ciphertext to the Dec oracle, when $b = 1$ we replace Dec with an oracle which simply returns $\perp$ and restrict the adversary from never querying the oracle with a previously seen ciphertext. The probability that the adversary distinguishes the two cases is precisely equal to the probability of winning the INT-CTXT game.

Hence it is trivial to show that AE-ROR security implies both IND-CPA and INT-CTXT security.

Note that although this is a convenient definition to use for authenticated encryption, current use of authenticated encryption (with associated data) takes additional inputs in the form of a nonce and header data. In practice, randomised schemes are constructed by using nonces, and thus this formulation helps cover nonce-misuse (i.e. repeated nonces) which is not necessarily covered by abstracting away this concern and treating Π as a randomised scheme. More details can be found in the original work by Rogaway and Shrimpton [170].

Furthermore, we omit details on any recent developments in the study of authenticated encryption, including, for example, tidiness [151] and length-regularity [24].

2.2.3 Pseudorandom Functions

One of the most fundamental primitives in use in cryptography is the *pseudorandom function* (PRF). In this thesis, we frequently make use of PRFs and variants thereof as a building block for constructing larger systems and protocols.

Definition 5 (Pseudorandom Function)**.** Let F be a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where we denote the restriction of F to a single $k \in \mathcal{K}$ by $F_k : \mathcal{X} \to \mathcal{Y}$.

We say that F is a *pseudorandom function* if k, chosen uniformly at random, F_k is indistinguishable from a function f chosen uniformly at random from the set of functions $f : \mathcal{X} \to \mathcal{Y}$.

Concretely, define the PRF-ROR game in which an adversary has access to a real-or-random PRF oracle RoR. When the hidden bit $b = 0$, the oracle returns $F_k(x)$, and when $b = 1$, the oracle returns $f(x)$, where $k \leftarrow_{\$} \mathcal{K}$ and $f \leftarrow_{\$} \{f : \mathcal{X} \to \mathcal{Y}\}$ are generated uniformly at random at the start of the game.

The adversary wins the PRF-ROR game if at the end of the game they return the correct bit $b' = b$.

We define the advantage of an adversary $\mathcal{A}$, making at most q queries, in attacking the pseudorandom function F to be

$$\mathsf{Adv}^{\text{prf-ror}}_{F,q}(\mathcal{A}) = 2 \cdot \Pr[\text{PRF-ROR} \Rightarrow \mathsf{true}] - 1.$$

PRFs are useful building blocks in many primitives. As a simple example, and one which will be used frequently in later chapters, we show how to build a symmetric encryption scheme from a PRF.

Definition 6. PRF-CTR encryption scheme Let F be a PRF, $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Then define the symmetric encryption scheme Π_F as follows:

- $\mathsf{KeyGen}()$: outputs a random key $k \leftarrow_{\$} \mathcal{K}$.

- $\mathsf{Enc}(k, m)$: on input $m \in \{0,1\}^{n\ell}$, write $m = (m_1, \ldots, m_\ell)$ for $m_i \in \{0,1\}^n$, and return $C = (m_1 \oplus F(k,1), m_2 \oplus F(k,2), \ldots, m_\ell \oplus F(k,\ell))$ where ℓ is written as an n-bit value.
- $\mathsf{Dec}(k, C)$: recovers m from blocks $m_i = C_i \oplus F(k,i)$ for $1 \leq i \leq \ell$.

Note that we restrict the plaintext space of Π_F to $\{0,1\}^{n\ell}$ with $\ell < 2^n$ for simplicity.

For this scheme, we show that confidentiality depends on the security of the PRF F, a standard result.

Theorem 1. *Let Π_F be the PRF-CTR encryption scheme as defined above.*

Then for any adversary $\mathcal{A}$ attacking Π_F in the IND\$-CPA *game, making only a single query of length at most 2^n blocks, there exists an adversary $\mathcal{B}$ in the* PRF-ROR *game such that*

$$\mathsf{Adv}^{\text{ind\$-cpa}}_{\Pi_F,q}(\mathcal{A}) = \mathsf{Adv}^{\text{prf-ror}}_{F,q}(\mathcal{B}).$$

Proof. We prove this theorem by directly constructing an adversary $\mathcal{B}$. $\mathcal{B}$ simulates the IND\$-CPA game for Π_F by using its RoR oracle as F in the computation of Enc. When $\mathcal{A}$ returns the guess b', $\mathcal{B}$ uses this as its guess in the PRF-ROR game.

Suppose the hidden bit in the PRF-ROR game is 0, then $\mathcal{B}$ returns encryptions of the form $\text{Enc}(m) = (m_0 \oplus F(k,0), \ldots, m_\ell \oplus F(k,\ell))$. This perfectly simulates the IND\$-CPA game with bit $b = 0$.

Alternatively, suppose the hidden bit in the PRF-ROR game is 1, then $\mathcal{B}$ returns encryptions of the form $\text{Enc}(m) = (m_0 \oplus f(0), \ldots, m_\ell \oplus f(\ell))$.

Given that the adversary only makes a single Enc query of length at most 2^n, f is queried at most once for each input value, and since f is a random function, the outputs of f are uniformly and independently random values. Therefore the challenge ciphertext is distributed identically to a randomly sampled string.

This shows that using the RoR oracle to simulate message encryptions precisely matches the IND\$-CPA game, and therefore the probability that $\mathcal{A}$ outputs the correct bit is precisely its advantage in the IND\$-CPA game. Since $\mathcal{B}$ outputs the same bit as $\mathcal{A}$, $\mathcal{B}$'s output is then correct whenever $\mathcal{A}$'s is, we may conclude that

$$\mathsf{Adv}^{\text{ind\$-cpa}}_{\Pi,q}(\mathcal{A}) = \mathsf{Adv}^{\text{prf-ror}}_{F,q}(\mathcal{B}).$$

□

This theorem has an important restriction: the security only holds for the encryption of a single message. This assumption was crucial to show that applying the PRF was equivalent to a one-time pad. In order to relax this requirement, there are a couple of options: we can either choose a random IV to start the counter, in which case the reduction would then need to include the probability of two intervals overlapping; or we can make the scheme stateful and continue incrementing the counter. The former results in a lossy reduction, and the latter implies significant implementation difficulties.

2.2.4 Random Oracle Model

The random oracle model (ROM) was introduced by Bellare and Rogaway [35] and has since become a widely used and accepted paradigm.

Definition 7 (Random Oracle [35])**.** A *random oracle* R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $R(x)$ uniformly and independently for every x.

In practice, the oracle R will be used to provide multiple generators $G : \{0,1\}^* \to \{0,1\}^\infty$ and random hash functions $H : \{0,1\}^* \to \{0,1\}^n$, where some appropriate encoding method is used to make each generator or function produce independent outputs. Of course, in practice protocols do not and cannot make use of infinite output, but rather the syntax $\{0,1\}^\infty$ represents the idea that any arbitrary length output can be produced.

The random oracle model (ROM) itself is not a concrete definition, but a common paradigm used to express the idea that a *well-chosen* hash function can fulfil the role of a random oracle. Using the ROM in proofs entails proving that a particular scheme is secure, assuming that a random oracle is made available to all parties (in particular, the adversary can freely query the random oracle). A concrete implementation of the scheme would then use a hash function instead of the random oracle.

Use of the ROM can be contentious. In [72], it was shown that there exist several carefully constructed cryptosystems which are secure in the random oracle model but completely broken when instantiated with any hash function.

These cryptosystems are very contrived. The intuition is that for every input x, the algorithm checks whether $(x, H(x))$ is in some set $\mathcal{S}$. If true, then it does something clearly insecure, e.g. returns the secret key. Suppose we let $\mathcal{S}$ be the set of all possible pairs of values $(x, h(x))$ for $x \in \{0,1\}^*$ and h from the set of all possible hash functions we are selecting from. Then if H is a random oracle, it is unlikely to satisfy that relation for any choice of h, However, clearly if H is from that set of hash functions, then $(x, H(x)) \in \mathcal{S}$ will be true.

Nonetheless, the existence of such constructions raises serious doubts about the legitimacy of the ROM. However, the ROM remains a widely used paradigm, with the caveat that care must be taken to ensure constructions do not rely on (the absence of) specific features of the underlying hash functions.

2.2.5 Diffie–Hellman

The Diffie–Hellman (DH) problem dates back to the very beginning of public-key cryptography [89]. Since then, many variants of the DH problem have been constructed as base assumptions for various cryptosystems.

Two of the most common DH variants are the computational Diffie–Hellman (CDH) and the decision Diffie–Hellman (DDH) problems.

Definition 8 (CDH Game)**.** Let $\mathbb{G}$ be a group of order p.

In the CDH game, an adversary is given a tuple (g, g^x, g^y) where g is a generator of $\mathbb{G}$ and x, y are sampled uniformly at random from $\mathbb{Z}_p$.

The adversary wins if they output a value h such that $h = g^{xy}$.

For an adversary $\mathcal{A}$, we define the advantage as

$$\mathsf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\mathcal{A}(g, g^x, g^y) = g^{xy}] .$$

Definition 9 (DDH Game)**.** Let $\mathbb{G}$ be a group of order p.

In the DDH game, a bit $b \leftarrow_{\$} \{0, 1\}$ is sampled uniformly at random, and the adversary is given the tuple (g, g^x, g^y, g^z) where g is a generator of $\mathbb{G}$ and x, y are sampled uniformly at random from $\mathbb{Z}_p$, and z is either xy, when $b = 0$, or another uniformly random exponent from $\mathbb{Z}_p$, when $b = 1$.

The adversary outputs a guess b' for the hidden bit, and the DDH game returns true if $b' = b$.

For an adversary $\mathcal{A}$, we define the advantage as

$$\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}) = 2 \cdot \Pr[\mathrm{DDH} \Rightarrow \mathsf{true}] - 1.$$

It can easily be shown that for all adversaries $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that $\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{B}) \geq \mathsf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A})$. Given a DDH sample (g, g^x, g^y, g^z), the adversary $\mathcal{B}$ uses $\mathcal{A}$ as an oracle on (g, g^x, g^y) to compute g^{xy} and returns $b = 0$ if $g^z = g^{xy}$, winning the DDH game whenever $\mathcal{A}$ is able to construct g^{xy} and hence wins in the CDH game.

However, the other direction is still unknown in general, and in certain Gap-DH groups DDH is easy, whereas CDH is still conjectured to be hard [118].

A summary of other known results for these two problems can be found in [58]. Currently, both problems are *assumed* to be difficult to solve in certain groups, and *conjectured* to be equivalent to solving the discrete logarithm problem. It is easy to see that solving the discrete logarithm problem (given $h = g^x$ find x) implies an efficient algorithm for both DDH and CDH.

We often state a priori, for example, "where $\mathbb{G}$ is a group in which the DDH assumption holds", and in the security reduction we will see the $\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}$ term present. However, care must be taken to consider the concrete security of this construction, as we would with any other reduction. For example, the attack on TLS in [6] performed just a single expensive pre-computation step to allow for solving the discrete logarithm problem in a fixed group, which could then be used to attack multiple subsequent connections.

As before, the concrete security level is contextual. In the previous example, the group size was 512-bits, which resulted in a pre-computation effort of approximately 10 years of CPU time (roughly 2^{60} operations). This could be seen as prohibitive to attack a single connection in real-time, but as an amortised effort to attack multiple future connections, suddenly the attack becomes very efficient.

This is just the beginning of hardness assumptions stemming from the DH problem, others include: the Gap-DH problem [153]; DH style assumptions in bilinear groups [59]; strong and oracle variants [1]; and one-more variants [56]. Finally, we refer the reader to [58] for more security properties and uses of DDH.

Applications

Although initially conceived as a mechanism for exchanging keys, the DH problem has become the core assumption in many cryptosystems and protocols.

Here, we consider two important uses of DH: authenticated key exchange, and constructing a PRF.

Station-to-Station Protocol

The station-to-station (STS) protocol was proposed by Diffie, van Oorschot, and Wiener [90] as a simple authenticated key exchange protocol, using the power of the DH problem and a public-key infrastructure (PKI) in order to exchange keys.

The intuition is as follows: suppose Alice and Bob send each other DH shares g^x and g^y respectively. Then the CDH problem states that the adversary cannot compute g^{xy}, and going further the DDH problem states that they cannot distinguish g^{xy} from a random value g^z. However, Alice and Bob can easily compute g^{xy} by using their private exponents to compute $(g^y)^x$ and $(g^x)^y$ respectively.

Therefore, against a *passive* adversary, this is a reasonable key exchange protocol, under the assumption that the DDH problem is hard. However, suppose the adver-

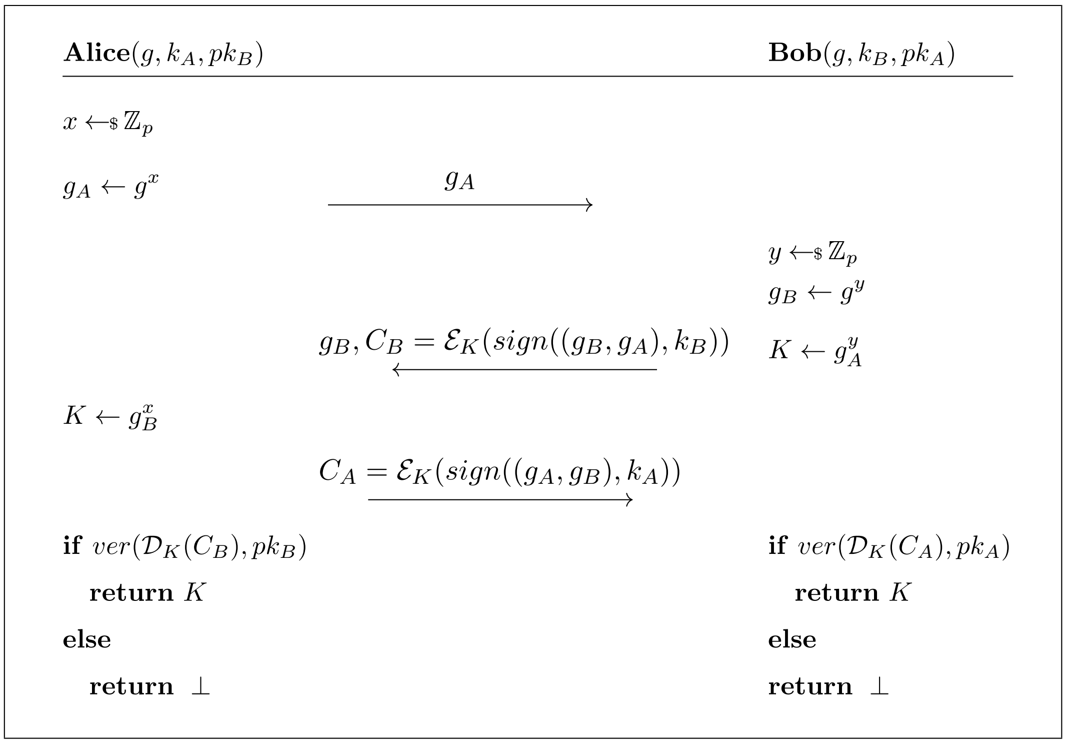
Alice(g, k_A, pk_B)
Bob(g, k_B, pk_A)
$x \leftarrow_{\$} \mathbb{Z}_p$
$g_A \leftarrow g^x$
g_A
$y \leftarrow_{\$} \mathbb{Z}_p$
$g_B \leftarrow g^y$
$g_B, C_B = \mathcal{E}_K(sign((g_B, g_A), k_B))$
$K \leftarrow g_A^y$
$K \leftarrow g_B^x$
$C_A = \mathcal{E}_K(sign((g_A, g_B), k_A))$
if $ver(\mathcal{D}_K(C_B), pk_B)$ **return** K **else** **return** $\perp$
if $ver(\mathcal{D}_K(C_A), pk_A)$ **return** K **else** **return** $\perp$

Figure 2.2: The basic station-to-station (STS) protocol.

sary plays the role of an active man-in-the-middle (MITM). Then it can relay values $g^{y'}$ to Alice and $g^{x'}$ to Bob, leading them to compute keys $g^{xy'}$ and $g^{x'y}$. At this point, the adversary can compute both keys and eavesdrop on any communication protected using keys derived from these exchanged values.

The STS protocol solves this by use of a PKI. Informally, when using a PKI, we assume there exists a mechanism to distribute long-term keys, bound to the participants' identities. Hence the STS protocol enhances the vanilla DH key exchange by exchanging certificates and signing appropriately. In the following, let k_A denote private signing keys, and pk_A denote public verification keys. We write $sign(x, k_A)$ to denote signing the message x using key k_A using some signature scheme (the details of which we omit for simplicity), and $ver(S, pk_A)$ verification of the signature S using the public key pk_A. Furthermore let g be a public generator of the group $\mathbb{G}$ or order p. The protocol is depicted in Figure 2.2.

PRF Construction in the Random Oracle Model

As mentioned in Section 2.2.3, we commonly make use of PRFs in our constructions and examples. One simple instantiation of a PRF comes from Naor, Pinkas and Reingold [152], and relies on the DDH assumption, and the random oracle model.

Definition 10 (DDH-ROM Pseudorandom Function)**.** Let $\mathbb{G}$ be a group of order p in which the DDH assumption holds, and let $H : \{0,1\}^* \to \mathbb{G}$ be a hash function modelled as a random oracle.

Then define the DDH-ROM PRF as $F : \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$ by $F(k, m) = H(m)^k$.

Theorem 2. *Let F be the DDH-ROM PRF as described previously. For any* PRF-ROR *adversary $\mathcal{A}$, making at most q random oracle and* RoR *queries, distinguishing F from a random function f, there exists an adversary $\mathcal{B}$ acting as a DDH distinguisher such that*

$$\mathsf{Adv}_F^{\text{prf-ror}}(\mathcal{A}) \leq q \cdot \mathsf{Adv}_{\mathbb{G}}^{\text{ddh}}(\mathcal{B}).$$

The following proof relies on the *programmability* of the random oracle used. This is a useful trick which allows us to set the random oracle for an input-output pair, provided the output is still distributed uniformly at random. For more information on this technique, see [106].

We give the proof in full as an introduction to the style of proofs used throuhgout this thesis, and as an expanded version of the proof sketch given in [152].

Proof. We prove this with a series of q hybrid game hops.

Each game starts by generating a random PRF key k.

In game G_0, on each query m_i to the random oracle H, generate a random value r_i and program the response as $H(m_i) := g^{r_i}$.

On each query m^* to RoR, look up the value of $H(m^*)$ programmed in H (generating a new entry if necessary), and return $(g^{r^*})^k$.

This perfectly simulates the PRF-ROR game for $\mathcal{A}$, with the hidden bit $b = 0$.

Game G_i is identical to game G_{i-1} for $1 \leq i \leq q+1$, except on calling the PRF oracle RoR for the i-th query value m_i, we replace the output with a randomly sampled value.

Finally, in game G_q, all outputs for RoR are generated uniformly at random, perfectly simulating the PRF-ROR game, with hidden bit set to $b = 1$.

The definition of the advantage of $\mathcal{A}$ in the PRF-ROR game is given by

$$\begin{aligned}\mathsf{Adv}_F^{\text{prf-ror}}(\mathcal{A}) &= |2 \cdot \Pr[\text{PRF-ROR} \Rightarrow \mathsf{true}] - 1| \\ &= |(\Pr[\mathcal{A} \Rightarrow 0 \mid b = 0] + \Pr[\mathcal{A} \Rightarrow 1 \mid b = 1]) - 1| \\ &= |\Pr[\mathcal{A} \Rightarrow 0 \mid b = 0] - \Pr[\mathcal{A} \Rightarrow 0 \mid b = 1]|.\end{aligned}$$

Let S_i represent the event that $\mathcal{A}$ outputs 0 in game G_i. From the above, we have that $\mathsf{Adv}_F^{\text{prf-ror}}(\mathcal{A}) = |\Pr[S_0] - \Pr[S_q]|$.

Using the triangle inequality, we can write:

$$\begin{aligned}|\Pr[S_0] - \Pr[S_q]| &= |\Pr[S_0] - \Pr[S_1] + \cdots - \Pr[S_{q-1}] + \Pr[S_{q-1}] - \Pr[S_q]| \\ &\leq |\Pr[S_0] - \Pr[S_1]| + \cdots + |\Pr[S_{q-1}] - \Pr[S_q]|.\end{aligned}$$

Next, we construct a series of adversaries $\mathcal{B}_i$ who simulate the environment for $\mathcal{A}$ in games G_i, G_{i+1} for all $0 \leq i \leq q-1$.

At the start of the game, $\mathcal{B}_i$ calls the DDH oracle for the query tuple (g^x, g^y, z). For messages m_j with $j \neq i$, the adversary as before programs the random oracle such that $H(m_j) = g^{r_k}$. Furthermore, RoR oracle queries are answered using $(g^y)^{r_k}$, using y as the implicit PRF key.

However, for the i-th query, the oracle is programmed as $H(m_i) = g^x$, and the RoR query is returned as z.

If the hidden bit in the DDH game is 0, then $z = g^{xy}$, and this models game G_i. On the other hand, if the hidden bit is 1, then z is a uniformly random value, which simulates game G_{i+1}.

When $\mathcal{A}$ returns a guess bit b', $\mathcal{B}_i$ will use this as its guess in the DDH game. As detailed above, $\mathcal{B}_i$ is correct when $\mathcal{A}$ outputs 0 in game G_i, and 1 in game G_{i+1}, which corresponds to the events S_i and $\neg S_{i+1}$ respectively.

Hence we get $\mathsf{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{B}_i) = |2 \cdot (\frac{1}{2}\Pr[S_i] + \frac{1}{2}(1 - \Pr[S_{i+1}])) - 1| = |\Pr[S_i] - \Pr[S_{i+1}]|$.

This argument was general for all i such that $0 \leq i \leq q-1$, and inserting into the previous formula, it can be seen that the sum $\sum_i \mathsf{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{B}_i)$ an upper bound of $\mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{A})$. Therefore there must exist at least one particular i such that

$$\mathsf{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{B}_i) \geq \frac{1}{q}\mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{A})$$

which can be rearranged to give the theorem statement. □

2.3 Symbolic Analysis

In computational analysis, one might consider the fundamental component to be the bitstring. In symbolic analysis, we instead treat variables as abstract quantities represented symbolically. Although in this approach we lose some granularity of proving – proof statements are binary as opposed to the quantitative nature of concrete security – this abstraction allows us to leverage powerful symbolic analysis tools to aid in producing proofs.

In recent years there have been a number of new tools created for modelling security protocols symbolically. Some examples are Scyther [83], ProVerif [50], and TAMARIN [176].

We note that there are also tools to produce automated proofs in the computational model, potentially offering the best of both approaches. One such example is CryptoVerif [52]. However, we do not cover these here and more can be found about these methods in [81] and [51].

In this section, we briefly introduce some core concepts of symbolic analysis, described in the context of the TAMARIN prover tool. This is not intended to be a comprehensive introduction to symbolic analysis, but rather to provide some insight into how such tools can be used to perform analysis of cryptographic protocols.

2.3.1 Tamarin Prover

The Tamarin prover [176] is a symbolic modelling and analysis tool for security protocols. Here we provide a brief introduction to Tamarin. For a more detailed introduction, we suggest reading the Tamarin manual found at [180], or [176, 175, 144, 113] for more details about both the formal underpinnings of Tamarin, and symbolic analysis in general.

The Tamarin semantics are based on multiset-rewriting. A Tamarin model defines a transition system whose state is a multiset of *facts*. The allowed transitions are specified by *rules*. At a very high level, Tamarin rules encode the behaviour of participants, as well as adversarial capabilities. In modelling cryptographic protocols, these rules play a role similar to oracles in Bellare-Rogaway style models.

Tamarin rules have a *left-hand side* (premises), *actions*, and a *right-hand side* (conclusions). The left-hand and right-hand sides of rules respectively contain multisets of facts. Facts can be *consumed* (when occurring in premises) and *produced* (when occurring in conclusions). Each fact can be either *linear* or *persistent*, the latter marked with an exclamation point. While linear facts model limited resources that cannot be consumed more times than they are produced, persistent facts model unlimited resources, which can be consumed any number of times once they have been produced.

A rule can only be executed if its left-hand side can be matched with facts that are available for consumption in the current state. For instance, the `Fresh` rule depicted here

```
rule Fresh:
    [ ]--[ ]->[ Fr(~x) ]
```

has no premises or actions, and every execution of it produces a single linear `Fr(~x)` fact. Note that only the `Fresh` rule can produce `Fr` facts, each of them unique. Use of the ~ symbol denotes a variable of the type `Fresh`. Other variable types include `Public`, denoted by $, and `Temporal`, denoted by #. Fresh values are the quanta of Tamarin, representing a unique discrete value. Any variable without any typing information can be any one of the above types, or a combination of terms when used within an equation. We often omit marking variables as fresh where it is either clear from the context, or unimportant for the exposition.

Actions do not influence the transitions, but are "logged" when rules are triggered as a means of incrementally constructing observable action traces that in turn represent a record of a specific execution. Actions (as part of action traces) form the glue between the defined transition system and the property specification language.

2.3.2 Cryptographic primitives

As mentioned in the introduction, the symbolic modelling approach considers all variables – inputs, outputs, keys, etc. – as abstract quantities, without a specific length, set, or distribution to be sampled from. This results in the assumption that cryptographic primitives are "perfect". For example, it is not possible to mount a brute force attack on encryption schemes to find the key, since the keyspace is effectively infinite.

To instantiate a cryptographic primitive symbolically, we define the functionality with a set of equations. Any potential flaws, weaknesses, or side-effects must be explicitly defined.

For example, a function F is by default assumed to take arbitrary length inputs, and return arbitrary length outputs. In the symbolic model variables are considered abstractly, and concepts such as the bit length are undefined. Without defining any additional information, this function F is a one-way function, since we have not defined a function F^{-1}.

Suppose we wish to model more properties about the output of these functions. We do so by adding equations which define their behaviour. For example, $\mathsf{Enc}(k, m)$ without any further information would be, as above, a one-way function, leaving m unrecoverable. Hence we must describe how decryption takes place. In TAMARIN, symmetric encryption is modelled by writing the equation describing its functionality:

```
sdec(senc(m, k), k) = m
```

where `k` is a shared secret key and `m` is a message.

This means that *symbolically*, the term `senc(m, k)` can be combined with `k` as an input to `sdec` to recover `m`.

Since `k` is an abstract variable, and the adversary can only learn information through explicit rules or equations, it is impossible for them to learn `k`, and thus they cannot compute `sdec(_, k)`. Therefore the adversary can never learn `m` (other than already knowing it beforehand).

Remark 1. The equation `sdec(senc(m, k), k) = m` for encryption is known as a *subterm convergent* equation since the term `m` appears as a value within the left-hand side. An example of an equation which is not subterm convergent is blinded signatures

```
unblind(sign(blind(m, r), k), r) = sign(m, k)
```

because the `sign(m, k)` does not appear anywhere on the left-hand side. Recent work has extended Tamarin to support such equations [97].

As certain primitives are used repeatedly across many cryptographic protocols, there are built-in definitions for them. The Tamarin *builtins* include equational theories for Diffie-Hellman group operations, asymmetric encryption, symmetric encryption, digital signatures and hashing. The theory required to support Diffie-Hellman is particularly interesting, and we cover it in more detail later.

The symmetric encryption builtin, for instance, could be used in this simple rule that models sending encrypted data out to a network:

```
rule Send:
    [Fr(~k), Fr(~data)]--[Send(~data)]->[Out(senc(~data,~k))]
```

The use of the builtin `Out` fact, as depicted in the `Send` rule, denotes that a message has been sent out to the network, i.e. `senc(~data,~k)` becomes known to the adversary. Receiving a message from the network is denoted by the corresponding `In` fact.

In other words, `In(senc(~data,~k))` could be a premise of a rule which models receiving encrypted data from the network.

2.3.3 Dolev-Yao Attacker

In the previous section we introduced `Out` and `In` as two builtin facts, used to model sending and receiving messages to and from the network respectively. We also stated that the message itself becomes available to the adversary. This is due to the fact that Tamarin (as is common in symbolic analysis) adheres to the Dolev-Yao attacker model of [94].

A Dolev-Yao attacker has complete control over the network. In particular, any message sent onto the network is given directly to the attacker, and the attacker is responsible for delivering messages to participants. In between, the attacker has complete freedom to disassemble and reassemble the message, change parts, or simply drop the message entirely.

A priori, participants have no way of knowing who the message was originally sent from, and to whom it was intended, since this information can be trivially forged by the adversary.

If we wish to represent the existence of a secure channel between parties, we would use regular facts, e.g. `AuthMessage($A, $B, m)` might represent securely sending the message `m` from `$A` to `$B`. Since `AuthMessage` would be an instance of a fact, these are directly passed between rules without being provided to the attacker. This is commonly used as a convenient way to check the model for correctness. By replacing all `In` and `Out` facts with `AuthMessage`, we can test the protocol in the absence of an attacker.

2.3.4 Properties and Proofs

In the computational approach of Section 2.2, security properties were quantified by the concrete advantage an adversary can achieve. However, in general there is no concept of probabilistic nor computationally bounded adversaries for symbolic tools. For example, in Tamarin results are binary. If a proof terminates, then the property is either validated or falsified.

As described by Blanchet in [51], security properties largely fall into two categories: trace and equivalence properties.

Trace properties state that a certain event should not happen. For example. in the INT-CTXT game, the adversary should not produce a valid, distinct ciphertext.

Equivalence properties, commonly known as indistinguishability in the computational setting, refer to the ability of an adversary to distinguish two views of a process or protocol. For example, in the IND-CPA game, the adversary is challenged to distinguish two worlds, one in which the LR oracle returns encryptions of m_0, and the other returning encryptions of m_1.

Currently, most symbolic analyses have focused on proving trace properties, which are better suited to the tools used. However, there has been some promising work in proving equivalence relationships. For example, the Tamarin tool has recently been extended to support proofs of equivalence [25].

Security properties as lemmas

Tamarin formulae are specified in a fragment of first-order logic and therefore offer the usual connectives (where `&` and `|` denote *and* and *or*, respectively), quantifiers `All` and `Ex`, and timepoint ordering `<`. In formulas, the prefix `#` denotes that the following variable is of type *timepoint*. The expression `Action(args)@t` denotes that `Action(args)` is logged in the action trace at point `t`, resulting from an instantiation of a rule.

We use the Tamarin property language to encode different kinds of properties as *lemmas*. These can include, for example, basic state reachability tests as well as security properties. We follow the common Tamarin modelling approach, in which lemmas closely resemble properties as defined in Bellare-Rogaway models. For instance, where a Bellare-Rogaway definition will typically restrict the set of oracle queries the adversary can make (e.g., it cannot query for a decryption of the challenge ciphertext), in Tamarin we restrict the adversary in the statement of the lemma.

A basic property that states the secrecy of `n` can be encoded as a simple lemma:

```
lemma Example_Property:
     "All A y n #i. Send(A, n, y) @ i ==> not Ex #j. K(n) @ j"
```

which says that for all actors A, sending message n using key y, at timepoint i, then there does not exist a timepoint j such that the adversary knows n. In other words, the adversary never learns n.

A more realistic lemma needs to account for the attacker model. For example, in the presence of an attacker who can compromise symmetric keys, a formula resembling `not Revealed(y) @ k` needs to be conjoined with the left-hand side of the above property or it will be trivially false.

IND-CPA **Security in Tamarin**

We use IND-CPA as an example of specifying an equivalence relation in Tamarin.

For example, we might define IND-CPA using the following setup:

```
rule LR_Oracle:
    [ !Key(k), Fr(r), In(m0), In(m1) ]
  --[ ]
  ->[ Out(senc(<diff(m0, m1), r>, k)) ]

rule Enc:
    [ !Key(k), Fr(r), In(m)]--[ ]->[ Out( senc(<m, r>, k) )]
```

Here the `diff(.,.)` term states that the two worlds created by selecting either of the two values are indistinguishable. In the above, we are using the Tamarin syntax `<_, _>` for tuples.

The above `LR_Oracle` rule specifies an oracle which takes as input a (persistent) key fact, a fresh value `r`, and two adversarially-chosen inputs `m0, m1`. It produces an encryption under the key `k`. Tamarin will effectively generate two models by using the `diff` keyword. One representing the left world, and one for the right. Usual properties and lemmas can be proved individually for each world. However, Tamarin will also generate a new lemma, called `lemma Observational_equivalence`, which we prove to show that the indistinguishability holds between the two worlds.

We also provide the adversary with a dedicated encryption oracle `Enc`.

In this simple example, proving the property would not be very interesting. Encryption is modelled as a perfect primitive, so the only way the adversary could distinguish the two would be learning the key k. Hence it is more common to prove secrecy properties by proving the adversary cannot learn the key k with a trace property.

INT-CTXT **Security in Tamarin**

Proving integrity of encryption has the same problems as the previous example. Modelling encryption as a perfect primitive means the adversary cannot perform bit manipulations to produce a forged ciphertext. However, it still gives an idea of syntactically how we would formulate such a property.

Given an adversary with encryption and decryption oracles:

```
rule Enc_Oracle:
    [ !Key(k), In(m), In(r) ]
  --[ Enc(k, m, senc(<m, r>, k)) ]
  ->[ Out(senc(<m, r>, k)) ]

rule Dec_Oracle:
    [ !Key(k), In(senc(<m, r>, k)) ]
  --[ Dec(k, senc(<m, r>, k)) ]
  ->[ Out(m)]
```

Then the trace property for integrity of ciphertexts might be written as:

```
lemma INT_CTXT:
    "All k c #i. Dec(k, c)@i
              ==> Ex m #j. Enc(k, m, c)@j"
```

The INT-CTXT property is specified by the lemma `INT_CTXT` which says for all keys `k` and ciphertexts `c` such that the *action* `Dec(k, c)` was invoked at timepoint `#i`, then there must have been a corresponding `Enc(k, m, c)` action at timepoint `#j`.

This states that the only way the adversary could submit a decryption query to the oracle is by having previously having queried for it from the encryption oracle.

This lemma would be trivially satisfied in any model which does not leak secret keys. Note that we are using the pattern-matching form of symmetric encryption, which instead of using the `sdec` equation for decryption requires that the input be an encrypted value. This is a subtle distinction which allows us to better model authenticated encryption.

Proof Traces

The verification algorithm of TAMARIN is based on constraint solving and multiset-rewriting techniques, which allows its users to prove intricate security properties in complex protocols exhibiting branches and loops. One of the distinguishing features of TAMARIN is its support for global mutable non-monotonic state. Put simply, non-monotonic state allows TAMARIN to handle ephemeral information accurately, and can model complex data structures such as databases. TAMARIN also includes an extensive graphical user interface that enables the visualisation and interactive construction of proofs.

These features make TAMARIN a good fit for the modelling and in-depth analysis of highly complex protocols such as TLS 1.3, which we cover in depth in Chapter 5.

In TAMARIN, a proof consists of showing that in *all possible traces*, the proposed statement holds true.

For example, in the previous `Example_Property`, TAMARIN would attempt to find a contradiction, that is, a state when `K(n)@j` and `Send(A, n, y)`. To achieve this, TAMARIN starts by considering all states at which `Send(A, n, y)` is true (this action may appear in multiple locations), and performs a backwards-search until either: it finds a contradiction, i.e. `K(n)@j`; the entire trace has been solved without a contradiction; or some other previously proven property is triggered.

The last case is particularly important. TAMARIN can use previous lemmas to prove the current property. This means that intermediate proofs can be constructed before proving the final property. Furthermore, the same idea is inherent in TAMARIN's support for inductive proofs. This approach to proving was fundamental in our TLS 1.3 work, detailed in Chapter 5.

There are two methods to construct proofs in TAMARIN. First of all, a heuristic-driven automated mode which efficiently guides the backwards search. On termination, this automated search will either provide a proof of the correctness of the statement, or a contradiction, representing an attack on the property. However, in complex protocols it is common that TAMARIN cannot necessarily produce a proof automatically. In this case, the user can use TAMARIN's interactive mode to guide the proof search. As the user acquires knowledge on proof strategies, these can then be turned into new lemmas, which may be possible to prove automatically, an approach also described in Chapter 5.

2.3.5 Authentication Properties

In 1997, Lowe [132] introduced a more granular set of authentication notions, ranging from simple conditions stating simply that protocol participants at some point were running the protocol, to full agreement on all data items.

These properties are particularly suitable for analysis using symbolic tools, and the syntax used by Lowe for describing the authentication properties (a process algebra, known as CSP) is close to the language used by TAMARIN.

Here, we give a few examples of Lowe's properties, as they could be expressed in TAMARIN. The weakest property specified is called *aliveness*:

Definition 11 (Aliveness [132])**.** We say that a protocol guarantees to an initiator A *aliveness* of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.

In TAMARIN, we might define this as follows:

```
lemma aliveness:
      "All A B #i. Complete(A, 'initiator', B)@i
            ==> Ex role peer #j. Running(B, role, peer)@j & #j < #i"
```

Note that although B is identified as running the protocol, their role and who they are communicating with (i.e. the peer) are left unspecified.

In comparison, a much stronger property is (non-injective) agreement:

Definition 12 (Agreement [132])**.** We say that a protocol guarantees to an initiator A *agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds.

Again, specified in TAMARIN, this might look like the following:

```
lemma agreement:
     "All A B ds #i. Complete(A, 'initiator', B, ds)@i
          ==> Ex #j. Running(B, 'responder', A, ds)@j & #j < #i"
```

Other properties specified by Lowe include:

Injective agreement: Same as agreement, but requires that the matching action (in the previous example, the `Running` action) is unique.

Full agreement: Agreement holds for *all* possible data items.

Recentness: Requires that the two participants are currently running the protocol. May be achieved through agreement on fresh information, enforcing overlapping protocol actions, or some form of timestamping.

2.3.6 Symbolic Diffie–Hellman

Earlier we described support for symbolic Diffie–Hellman as a particularly important feature of TAMARIN. Indeed, Diffie–Hellman is an essential part of many cryptographic protocols. TAMARIN support for DH was introduced in [176], and full details of the formal theory can be found there.

Symbolically defining mathematical objects can be seen as a straightforward extension of the way basic functionality is defined. For example, we define algebraic groups by specifying equations encapsulating the group axioms:

- Associativity: $x * (y * z) \simeq (x * y) * z$
- Commutativity: $x * y \simeq y * x$
- Identity: $x * 1 \simeq x$
- Inverse: $x * x^{-1} \simeq 1$, $(x^{-1})^{-1} \simeq x$

Similarly, DH groups are defined in Tamarin by defining the exponentiation operator $(g, x) \mapsto g^x$, and defining the set of exponents to be a multiplicative group as above. Currently, Tamarin does not support addition in the exponent group, and thus neither is it defined to multiply two group elements.

With this limited set of equations, we get the discrete logarithm and CDH assumptions for free: there is no defined method to take g^x to x, nor to derive g^{x*y} from g^x, g^y.

We need to take this a step further in order to make it inherently useful for Tamarin. To use the CDH assumption effectively within a backwards-search strategy requires the ability to deduce something like "if the adversary knows g^{xy}, then they must also know either x or y". This would allow us to directly prove that the adversary cannot learn a negotiated DH key without knowing either one of the private keys.

However, this is not necessarily the case, due to the existence of inverses. Suppose $y = x^{-1}$, then the adversary clearly knows g^{xy} by just knowing g. Consider that this can be made arbitrarily complex, by instead setting $y = x^{-1} * z$ and the adversary instead knows z.

Addressing this is subtle and complex, the details of which can be found in [176]. The intuition is that an element can be reduced using a finite set of rules due to Lankford [181, 114], using the technique of [79]. This by itself is not sufficient, and Tamarin additionally uses a number of heuristics and constraints in order to ensure that this process indeed results in a terminating algorithm.

2.3.7 Strengths and Weaknesses of Symbolic Analysis

Despite the fact that we have introduced a number of limitations with symbolic tools, specifically with the Tamarin tool, none of these limitations are inherent weaknesses of the approach, but rather an obstacle to overcome. Consider that Tamarin is constantly evolving. In its approximately 5-year lifetime, it has added rich support for Diffie-Hellman equations [176], indistinguishability proofs in the form of observational equivalence [25], and the removing the restriction of subterm convergent equations [97] opens up new possibilities.

As the tools improve, the limitations of symbolic analysis decreases and the benefits become more and more apparent. Here we argue that symbolic analysis in its current state is already a hugely beneficial tool for real-world protocols.

Real-world protocols are often constrained by multiple external factors, from compatibility with legacy systems, to engineering and efficiency goals. Due to this, it is rare that a cryptographic protocol as initially conceived and analysed is the same when ultimately specified and deployed in practice. As the protocol moves further away from the initial design, and adds more functionality, components, and options, it becomes harder to analyse the interaction of all of these parts.

Traditional pen and paper proofs rely on analysing components in isolation, and then manually composing each part together. This can be an arduous task. Further exacerbating this issue is that real-world protocols are often moving targets. One particularly relevant example is our work in Chapter 5 focused on the TLS 1.3 protocol specification drafts. Currently numbering 21 drafts, always staying up-to-date with TLS 1.3 is a tough proposition.

Symbolic tools can be used to produce automated analyses, and thus help to mitigate some of the above pain points. As evidenced by our work on TLS 1.3 in Chapter 5, Tamarin is capable of handling a large, complex protocol and all of its modes in composition. Furthermore, large parts of our proving process for TLS 1.3 was automated. This meant that when making changes to the model, we were able to perform a significant portion of the analysis automatically.

Tamarin is still a relatively new tool, and tool primarily used by academics for research, it is missing a number of quality-of-life features. For example, in developing our complex model of TLS 1.3, we relied on macros provided by an external preprocessor to make the model more robust and modular. However, these are not intrinsic limitations of the tool.

On the other hand, a fundamental limitation of symbolic analysis tools is the inability to offer the granular, concrete quantification of security levels we obtain using computational analysis. This prevents symbolic tools from finding certain classes of attacks.

Additionally, although one strength of symbolic analysis is the ability to reason about protocols comprehensively, our TLS 1.3 analysis pushes the limits of Tamarin. Simple configuration options, or conditional paths, generally have the impact in doubling the complexity of possible traces to be configured. This inhibits performing even larger, more complex analyses (for example, combining TLS 1.2 and TLS 1.3 into a single model). A potentially significant area of research would be to build support for compositional proofs within Tamarin itself. Hence, removing the necessity to manually stitch proofs together, and permitting more complex analyses.

Ultimately, the strength of Tamarin and other symbolic analyses is to complement the computational approach. In particular, provable security can provide us with a precise quantification of the security of certain primitives, and their use within simple protocols. We can then move to the symbolic setting, to consider the use of these primitives within a wider scope.

As symbolic analysis for cryptographic protocols matures, a potentially exciting area of research could be looking at how to better express the failure modes of cryptographic primitives symbolically, so that the concrete security bounds provided by the computational approach can be directly mapped onto the results proven by symbolic analysis.

2.4 Summary

This chapter introduced a variety of analysis techniques, both computational and symbolic, and gave examples of how these can be used to analyse some simple constructions.

We introduced techniques known collectively as practice-oriented provable security for computational analysis suitable for analysing real-world protocols. The focus of this methodology is to produce an explicit quantification of the security of a protocol, including the resources required by the hypothetical attacker and the reliance on other assumptions and primitives.

In addition, we introduced symbolic analysis. In comparison with provable security, symbolic analysis offers coarser results about the security of a system, usually a simple binary answer. However, the extensive tooling produced to support symbolic analysis offers the potential to automate significant parts of an analyses, and may be particularly suitable for analysing complex, changing specifications.

In the rest of this thesis, we give examples of applying these two methodologies to real-world scenarios, and showcase the benefits of both approaches.

Chapter 3

The Pythia PRF Service

Contents

In this chapter, we detail and investigate the security and applications of a new cryptographic primitive called a verifiable partially-oblivious PRF. We propose a modern PRF service called Pythia *designed to offer a level of flexibility, security and ease-of-deployability lacking in prior approaches.*

We give new formal definitions for the partially-oblivious PRF, and give proofs of security for Pythia *under these definitions. We also give a construction that additionally supports efficient bulk rotation of previously obtained PRF values to new keys. Performance measurements show that our construction, which relies on bilinear pairings and zero-knowledge proofs, is highly practical.*

3.1 Introduction

Security improves in a number of settings when applications can make use of a cryptographic key stored on a remote system. As an important example, consider the compromise of enterprise password databases. Best practice dictates that passwords be hashed and salted before storage, but attackers can still mount highly effective brute-force cracking attacks against stolen databases.

Well-resourced enterprises such as Facebook [149] have therefore incorporated remote cryptographic operations to harden password databases. Before a password is stored or verified, it is sent to a *PRF service* external to the database. The PRF service applies a cryptographic function such as HMAC to client-selected inputs under a service-held secret key. Barring compromise of the PRF service, its use ensures that stolen password hashes (due to web server compromise) cannot be cracked using an offline brute-force attack: an attacker must query the PRF service from a compromised server for each password guess. Such online cracking attempts can be monitored for anomalous volumes or patterns of access and throttled as needed.

While PRF services offer compelling security improvements, they are not without problems. Even large organisations can implement them incorrectly. For example, Adobe hardened passwords using 3DES but in ECB mode instead of CBC-MAC (or another secure PRF construction) [98], a poor choice that resulted in disclosure of many of its customers' passwords after a breach. Perhaps more fundamental is that existing PRF services do not offer graceful remediation if a compromise is detected by a client. Ideally it should be possible to cryptographically erase (i.e., render useless via key deletion) any PRF values previously used by the client, without requiring action by end users and without affecting other clients. In general, PRF services are so inaccessible and cumbersome today that their use is unfortunately rare.

In this section, we present a next-generation PRF service called Pythia to democratise cryptographic hardening. Pythia can be deployed within an enterprise to solve the issues mentioned above, but also as a public, multi-tenant web service suitable for use by any type of organisation or even individuals. Pythia offers several security features absent in today's conventional PRF services that are critical to achieving the scaling and flexibility required to simultaneously support a variety of clients and applications. As we now explain, achieving these features necessitated innovations in both cryptographic primitive design and system architecture.

Key features and challenges. We refer to an entity using Pythia as a *client*. For example, a client might be a web server that performs password-based authentication for all of its end users. Intuitively, PYTHIA allows such a client to query the service and obtain the PRF output $Y = F_k(t, m)$ for a message m and a tweak t of the client's choosing under a client-specific secret key k held by the service. Here, the tweak t is typically a unique identifier for an end user (e.g., a random salt). In our running password storage example, the web server stores Y in a database to authenticate subsequent logins.

PYTHIA offers security features that at first glance sound mutually exclusive. First, PYTHIA achieves message privacy for m while requiring clients to reveal t to the server. Message privacy ensures that the PRF service obtains no information about the message m; in our password-storage example, m is a user's password. At the same time, though, by revealing t to the PRF service, the service can perform fine-grained monitoring of related requests: a high volume or otherwise anomalous pattern of queries on the same t would in our running example be indicative of an ongoing brute-force attack and might trigger throttling by the PRF service.

By using a unique secret key k for each client, PYTHIA supports individual key rotation should the value Y be stolen (or feared to be stolen). With traditional PRF services and password storage, such key rotation is a headache, and in many settings impractical, because it requires transitioning stored values $Y_1, \ldots, Y_n$ (one for each user account) to a new PRF key. The only way to do so previously was to have all n users re-enter or reset their passwords. In contrast, the new primitive employed for F_k in PYTHIA supports fast key rotation: the server can erase k, replace it with a new key k', and issue a compact (constant-sized) token with which the client can quickly update all of its PRF outputs. This feature also enables forward-security in the sense that the client can proactively rotate k without disrupting its operation.

PYTHIA provides other features as well, but we defer their discussion to Section 3.2. Already, those listed above surface some of the challenging cryptographic tensions that PYTHIA resolves. For example, the most obvious primitive on which to base PYTHIA is an oblivious PRF (OPRF) [108], which provides message privacy. But for rate-limiting, PYTHIA requires clients to reveal t, and existing OPRFs cannot hide only a portion of a PRF input. Additionally, the most efficient OPRFs (c.f., [117]) are not amenable to key rotation. We discuss at length other related concepts (of which there are many) in Section 3.9.

Partially-oblivious PRFs. We introduce *partially oblivious PRFs* (PO-PRFs) to rectify the above tension between fine-grained key management and bulk key management and achieve a primitive that supports batch key rotation. We give a PO-PRF protocol in the random oracle model (ROM) similar to the core of the identity-based non-interactive key exchange protocol of Sakai, Ohgishi, and Kasahara [174]. This same construction was also considered as a left-or-right constrained PRF by Boneh and Waters [63]. That said, the functionality achieved by our PO-PRF is distinct from these prior works and new security analyses are required. Despite relying on pairings, we show that the full primitive is fast even in our prototype implementation.

In addition to a lack of well-matched cryptographic primitives, we find no supporting formal definitions that can be adapted for verifiable PO-PRFs. (Briefly, previous definitions and proofs for fast OPRFs rely on hashing in the ROM before outputting a value [70, 117]; in our setting, hashing breaks key rotation.) We propose a new assumption (a one-more bilinear decisional Diffie-Hellman assumption), give suitable security definitions, and prove the security of the core primitive in Pythia under these definitions. Our new definitions and technical approaches may be of independent interest.

Using Pythia in applications. We implement Pythia and show that it offers highly practical performance on Amazon EC2 instances. Our experiments demonstrate that Pythia is practical to deploy using off-the-shelf components, with combined computation cost of client and server under 12 milliseconds. A single 8-core virtualised server can comfortably support over 1,000 requests per second, which is already within a factor of two of a standard HTTPS server in the same configuration. (Our Pythia implementation performs all communication over TLS.) We discuss scaling to handle more traffic volume later in this chapter; it is straightforward given current techniques.

We demonstrate the benefits and practicality of Pythia for use in a diverse set of applications. First is our running example above: we build a new password-database system using a password "onion" that combines parallelised calls to Pythia and a conventional key hashing mechanism. Our onion supports Pythia key rotation, hides back-end latency to Pythia during logins (which is particularly important when accessing PYTHIA as a remote third-party service), and achieves high security in a number of compromise scenarios.

Finally, we show that PYTHIA provides valuable features for different settings apart from enterprise password storage. We implement a client that hardens a type of password-protected virtual-currency account called a “brainwallet” [64]; use of PYTHIA here prevents offline brute-force attacks of the type that have been common in Bitcoin.

Our prototype implementation of PYTHIA is built with open-source components and itself is open-source. We have also released Amazon EC2 images to allow companies, individuals, and researchers to spin-up PYTHIA instances for experimentation.

3.2 Overview and Challenges

We now give a high-level overview of PYTHIA, the motivations for its features, what prior approaches we investigated, and the threat models we assume. First we fix some terminology and a high-level conceptual view of what a PRF service would ideally provide. The service is provisioned with a master secret key msk. This will be used to build a tree that represents derived sub-keys and, finally, output values. See Figure 3.1, which depicts an example derivation tree associated with PYTHIA as well as which portions of the tree are held by the server (within the large box) and which are held by the client (the leaves). Keys of various kinds are denoted by circles and inputs by squares.

From the msk we derive a number of *ensemble keys*. Each ensemble key is used by a client for a set of related PRF invocations — the ensemble keys give rise to isolated PRF instances. We label each ensemble key in the diagram by $\mathtt{K}[w]$. Here w indicates a client-chosen *ensemble selector*. An *ensemble pre-key* $\mathtt{K}[w]$ is a large random value chosen and held by the server. Together, msk and $\mathtt{K}[w]$ are used to derive the ensemble key $k_w = \text{HMAC}(msk, \mathtt{K}[w])$. A table is necessary to support cryptographic erasure of (or updates to) individual ensemble keys, which amounts to deleting (or updating) a table entry.

Each ensemble key can be used to obtain PRF outputs of the form $F_{k_w}(t, m)$ where F is a (to-be-defined) PRF keyed by k_w, and the input is split into two parts. We

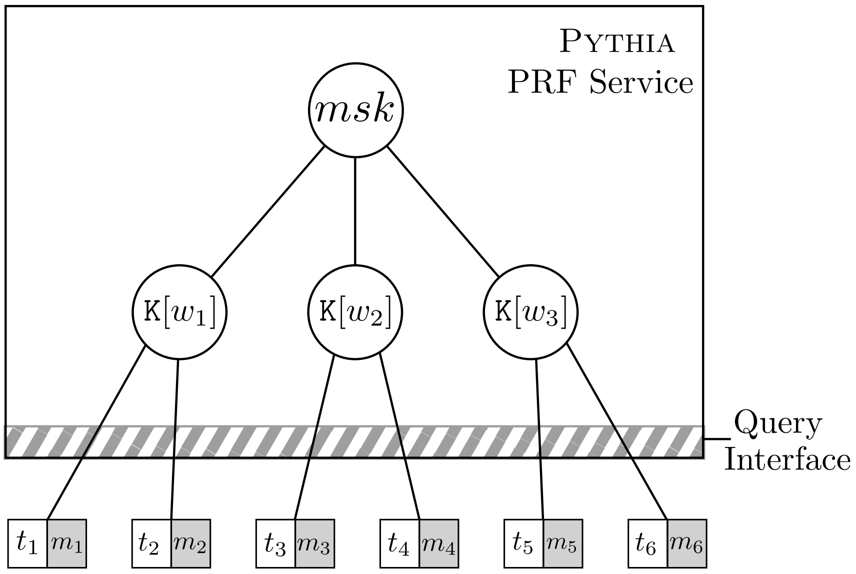

Figure 3.1: Diagram of PRF derivations enabled by PYTHIA. Everything inside the large box is operated by the server, which only learns tweaks and not the shaded messages.

call t a *tweak* following [130] and m the message. Looking ahead t will be made public to PYTHIA while m will be private. This is indicated by the shading of the PRF output boxes in the figure.

Deployment scenarios. To motivate our design choices and security goals, we relay several envisioned deployment scenarios for PYTHIA.

Enterprise deployment: A single enterprise can deploy PYTHIA internally, giving query access only to other systems they control. A typical setup is that PYTHIA fields queries from web servers and other public-facing systems that are, unfortunately, at high risk of compromise. PRF queries to PYTHIA harden values stored on these vulnerable servers. This is particularly suited to storing check-values for passwords or other low-entropy authentication tokens, where one can store $F_{k_w}(t, m)$ where t is a randomly chosen, per-user identifier (a salt) and m is the low-entropy password or authentication token. Here w can be distinct for each server using PYTHIA.

Public cloud service: A public cloud such as Amazon EC2, Google Compute Engine, or Microsoft Azure can deploy PYTHIA as an internal, multi-tenant service for their customers. Multi-tenant here means that different customers query the same PYTHIA service, and the cloud provider manages the service, ensemble pre-key table, etc. This enables smaller organisations to obtain the benefits of using Pythia for other cloud properties (e.g., web servers running on virtual machine instances) while leaving management of Pythia itself to experts.

Public Internet service: One can take the public cloud service deployment to the extreme and run Pythia instances that can be used from anywhere on the Internet. This raises additional performance concerns, as one cannot rely on fast intra-datacenter network latencies (sub-millisecond) but rather on wide-area latencies (tens of milliseconds). The benefit is that Pythia could then be used by arbitrary web clients, for example we will explore this scenario in the context of hardening brainwallets via Pythia.

One could tailor a PRF service to each of these settings, however it is better to design a single, application-agnostic service that supports all of these settings simultaneously. A single design permits reuse of open-source implementations; standardised, secure-by-default configurations; and simplifies the landscape of PRF services.

Security and functionality goals. Providing a single suitable design requires balancing a number of security and functionality goals. The most obvious requirements are for a service that: provides low-latency protocols (i.e., single round-trip and amenable for implementation as simple web interfaces); scales to hundreds of millions of ensembles; and produces outputs indistinguishable from random values even when adversaries can query the service. To this list of basic requirements we add:

- *Message privacy*: The PRF service must learn nothing about m. Message privacy supports clients that require sensitive values such as passwords to remain private even if the service is compromised, or to promote psychological acceptability in the case that a separate organisation (e.g., a cloud provider) manages the service.
- *Tweak visibility*: The server must learn tweak t to permit fine-grained rate-limiting of requests[1]. In the password storage example, a distinct tweak is

[1]In principle, the server need only be able to link requests involving the same t, not learn t. Explicit presentation of t is the simplest mechanism that satisfies this requirement.

assigned to each user account, allowing the service to detect and limit guessing attempts against individual user accounts.

- *Verifiability:* A client must be able to verify that a PRF service has correctly computed F_{k_w} for an ensemble selector w and tweak/message pair t, m. This ensures, after first use of an ensemble by a client, that a subsequently compromised server cannot surreptitiously reply to PRF queries with incorrect values. This matters, for example, if an attacker compromises the communication channel but not the server's secrets (msk and $\mathtt{K}[w]$). Such an attacker must not be able to convince the client that arbitrary or incorrect values are correct.

- *Client-requested ensemble key rotations:* A client must be permitted to request a rotation of its ensemble pre-key $\mathtt{K}[w]$ to a new one $\widehat{\mathtt{K}[w]}$. The server must be able to provide an update token Δ_w to roll forward PRF outputs under $\mathtt{K}[w]$ to become PRF outputs under $\widehat{\mathtt{K}[w]}$, meaning that the PRF is *key-updatable* with respect to ensemble keys. Additionally, Δ_w must be *compact*, i.e., constant in the number of PRF invocations already performed under w. Clients can mandate that rotation requests be authenticated (to prevent malicious key deletion). A client must additionally be able to *transfer* an ensemble from one selector w to another selector w'.

- *Master secret rotations:* The server must be able to rotate the master secret key msk with minimal impact on clients. Specifically, the PRF must be key-updatable with respect to the master secret key msk so that PRF outputs under msk can be rolled forward to a new master secret $\widehat{msk}$. When such a rotation occurs, the server must provide a compact update token δ_w for each ensemble w.

- *Forward security:* Rotation of an ensemble key or master secret key results in complete erasure of the old key and the update token.

Two sets of challenges arise in designing Pythia. The first is cryptographic. It turns out that the combination of requirements above are not satisfied by any existing protocols we could find. Ultimately we realised a new type of cryptographic primitive was needed that proves to be a slight variant of oblivious PRFs and blind signatures. We discuss the new primitive, and our efficient protocol realising it, in the next section. The second set of challenges surrounds building a full-featured service that provides the core cryptographic protocol, which we treat in Section 3.4.

3.3 Partially-oblivious PRFs

We introduce the notion of a (verifiable) partially-oblivious PRF. This is a two-party protocol that allows the secure computation of $F_{k_w}(t, m)$, where F is a PRF with server-held key k_w and t, m are the input values. The client can verify the correctness of $F_{k_w}(t, m)$ relative to a public key associated to k_w. Following our terminology, t is a tweak and m is a message. We say the PRF is partially oblivious because t is revealed to the server, but m is hidden from the server.

Partially oblivious PRFs are closely related to, but distinct from, a number of existing primitives. A standard oblivious PRF [108], or its verifiable version [117], would hide both t and m, but masking both prevents granular rate limiting by the server. Partially blind signatures [2] allow a client to obtain a signature on a similarly partially blinded input, but these signatures are randomised and the analysis is only for unforgeability which is insufficient for security in all of our applications.

We provide more comparisons with related work in Section 3.9 and a formal definition of the new primitive in Section 3.6. Here we will present the protocol that suffices for Pythia. It uses an admissible bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ over groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order q, and a pair of hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \{0,1\}^* \to \mathbb{G}_2$ (that we will model as random oracles). More details on pairings are provided in Section 3.6. A secret key k_w is an element of $\mathbb{Z}_p$. The PRF F that the protocol computes is:

$$F_{k_w}(t, m) = e\big(H_1(t), H_2(m)\big)^{k_w} .$$

This construction coincides with the Sakai, Ohgishi, and Kasahara [174] construction for non-interactive identity-based key exchange, where t and m would be different identities and k_w a secret held by a trusted key authority. Likewise, this construction is equivalent to the left-or-right constrained PRF of Boneh and Waters [63]. The contexts of these prior works are distinct from ours and our analyses will neces-

```
PRF-Cl($w, t, m$)                                          PRF-Srv($msk$)
$r \leftarrow_{\$} \mathbb{Z}_q$
$x \leftarrow H_2(m)^r$            $\xrightarrow{w, t, x}$
                                                           $\tilde{x} \leftarrow e(H_1(t), x)$
                                                           $k_w \leftarrow \mathrm{HMAC}(msk, \mathtt{K}[w])$
                                                           $p_w \leftarrow g^{k_w}$
                                                           $y \leftarrow \tilde{x}^{k_w}$
                                   $\xleftarrow{p_w, y, \pi}$    $\pi \leftarrow_{\$} \mathrm{ZKP}(\mathrm{DL}_g(p_w) = \mathrm{DL}_{\tilde{x}}(y))$
if $p_w$ matches $\wedge\, \pi$ verifies then
  return $y^{1/r}$
else
  return $\perp$
```

Figure 3.2: The partially-oblivious PRF protocol used in Pythia. The value π is a non-interactive zero-knowledge proof that the indicated discrete logs match. The client also checks that p_w matches ones seen previously when using selector w.

sarily be different, but we note that all three settings similarly exploit the algebraic structure of the bilinear pairing.

The client-server protocol that computes $F_{k_w}(t, m)$ in a partially-oblivious manner is given in Figure 3.2. There we let g be a generator of $\mathbb{G}_1$. We now explain how the protocol achieves our requirements described in the last section.

Blinding the message: In our protocol, the client blinds the message m, hiding it from the server, by raising it to a randomly selected exponent $r \leftarrow_{\$} \mathbb{Z}_q$. As $e\big(H_1(t), H_2(m)^r\big) = e\big(H_1(t), H_2(m)\big)^r$, the client can unblind the output y of PRF-Srv by raising it to $1/r$. This protocol hides m unconditionally, as $H_2(m)^r$ is a uniformly random element of $\mathbb{G}_2$.

Verifiability: The protocol enables a client to verify that the output of PRF-Srv is correct, assuming the client has previously stored p_w. The server accompanies the output y of the PRF with a zero-knowledge proof π of correctness.

PRF-Cl(w, t, m)		PRF-Srv(msk)
	$\xrightarrow{w, t, m}$	
		$\boldsymbol{\tilde{x} \leftarrow H_3(t \| m)}$
		$k_w \leftarrow \mathrm{HMAC}(msk, \mathtt{K}[w])$
		$p_w \leftarrow g^{k_w}$
		$y \leftarrow \tilde{x}^{k_w}$
	$\xleftarrow{p_w, y, \pi}$	$\pi \leftarrow\!\!\$\ \mathrm{ZKP}(\mathrm{DL}_g(p_w) = \mathrm{DL}_{\tilde{x}}(y))$
if p_w matches $\wedge\, \pi$ verifies **then**		
return $\boldsymbol{y}$		
else return $\perp$		

Figure 3.3: The unblinded PRF protocol supported by Pythia. Differences from the partially-oblivious protocol in Figure 3.2 are shown in **bold**.

Specifically, for a public key $p_w = g^{k_w}$, where g is a generator of $\mathbb{G}_1$, the server proves $\mathrm{DL}_g(p_w) = \mathrm{DL}_{\tilde{x}}(y)$. Standard techniques (see, e.g., Camenisch and Stadler [71]) permit efficient zero-knowledge proofs of this kind in the random oracle model. For example: the prover picks $v \leftarrow\!\!\$\ \mathbb{Z}_q$ and then computes $t_1 = g^v$ and $t_2 = \tilde{x}^v$ and $c \leftarrow H_3(g, p_w, \tilde{x}, y, t_1, t_2)$. Let $u = v - c \cdots k$. The proof is $\pi = (c, u)$. The verifier computes $t_1' = g^u \cdot p_w^c$ and $t_2' = \tilde{x}^u y^c$. It outputs true if $c = H_3(g, p_w, \tilde{x}, y, t_1', t_2')$. The notable computational costs for the server are one pairing and one exponentiation in $\mathbb{G}_T$; for the client, one pairing and two exponentiations in $\mathbb{G}_T$. Furthermore, the client's pairing can be pre-computed while waiting for the server's reply.

Efficient key updates: The server can quickly and easily update the key k_w for a given ensemble selector w by replacing the table entry $s = \mathtt{K}[w]$ with a new, randomly selected value s', thereby changing $k_w = \mathrm{HMAC}(msk, s)$ to $k_w' = \mathrm{HMAC}(msk, s')$. It can then transmit to the client an update token of the form $\Delta_w = k_w'/k_w \in \mathbb{Z}_q$.

The client can update any stored PRF value $F_{k_w}(t, m) = e\big(H_1(t), H_2(m)\big)^{k_w}$ by raising it to Δ_w; it is easy to see that $F_{k_w}(t, m)^{\Delta_w} = F_{k_w'}(t, m)$.

The server can use the same mechanism to update msk, which requires generating a new update token for each w and pushing these tokens to clients as needed.

Unblinded variants. For deployments where obliviousness of messages is unnecessary, we can use a faster, unblinded variant of the PYTHIA protocol that dispenses with pairings shown in Figure 3.3.

The only changes are that the client sends m to the server, there is no unblinding of the server's response, and, instead of computing

$$\tilde{x} \leftarrow e(H_1(t), x)$$

the server computes

$$\tilde{x} \leftarrow H_3(t \| m) \ .$$

All group operations in this unblinded variant are over a standard elliptic curve group $\mathbb{G} = \langle g \rangle$ of order q and we use a hash function $H_3 : \{0,1\}^* \rightarrow \mathbb{G}$.

An alternative unblinded construction would be to have the server apply the Boneh-Lynn-Shacham short signatures [62] to the client-submitted $t\|m$; verification of correctness can be done using the signature verification routine, and we can thereby avoid ZKPs. This BLS variant may save a small amount of bandwidth.

These unblinded variants provide the same services (verifiability and efficient key updates) and security with the obvious exception of the secrecy of the message m. In some deployment contexts an unblinded protocol may be sufficient, for example when the client can maintain state and submit a salted hash m instead of m directly. In this context, the salt should be held as a secret on the client and never sent to the server.

3.4 The Pythia Service Design

Table 3.1 gives the high-level API exposed by PYTHIA to a client. We now describe its functions in terms of the lifecycle of an ensemble key. We assume a security parameter λ specifying symmetric key lengths; a typical choice would be $\lambda = 128$.

We defer to later sections the underlying client-server protocols and to Section 3.5 details on key lifecycle management options, additional API calls for token management and ensemble transfer, and a discussion of master secret key rotation.

Command	**Description**
$\mathsf{Init}(w\,[, \mathit{options}])$	Create table entry $\mathtt{K}[w]$ (for ensemble key k_w)
$\mathsf{Eval}(w, t, m)$	Return PRF output $F_{k_w}(t, m)$
$\mathsf{Reset}(w,\ \mathit{authtoken})$	Update $\mathtt{K}[w]$ (and thus k_w); return update token Δ_w
$\mathsf{GetAuth}(w)$	Send one-time authentication token *authtoken* to client

Table 3.1: The basic Pythia API.

Ensemble initialisation. To begin using the Pythia service, a client creates an ensemble key for selector w by invoking $\mathsf{Init}(w\,[, \mathit{options}])$. Pythia generates a fresh, random table entry $\mathtt{K}[w]$. Recall that ensemble key $k_w = \text{HMAC}(msk, \mathtt{K}[w])$. So Init creates k_w as a byproduct.

Ideally, w should be an unguessable byte string. (An easily guessed one may allow attackers to squat on a key selector, thereby mounting a denial-of-service (DoS) attack.) For some applications, as we explain below, this is not always possible. If an ensemble key for w already exists, then the Pythia service returns an error to the client. Otherwise, the client receives a message signifying that initialisation is successful.

Init includes a number of options we detail in Section 3.5.

PRF evaluation. To obtain a PRF value, a client can perform an evaluation query $\mathsf{Eval}(w, t, m)$, which returns $F_{k_w}(t, m)$. Here t is a tweak and m is a message. To compute the PRF output, the client and server perform a one-round cryptographic protocol (meaning a single message from client to server, and one message back). We present details in Section 3.3, but remind the reader that t is visible to the server in the client-server protocol invoked by Eval, while m is blinded.

The server rate-limits requests based on the tweak t, and can also raise an alert if the rate limit is exceeded. We give example rate limiting policies in Section 3.7.

Ensemble-key reset. A client can request that an ensemble key k_w be reset by invoking $\mathsf{Reset}(w)$. This reset is accomplished by overwriting $\mathtt{K}[w]$ with a fresh, random value. The name service returns a compact (e.g., 256-bit) update token Δ_w that the client may use to update all PRF outputs for the ensemble. It stores this token locally, encrypted under a public key specified by the client, as explained below.

Note that reset *results in erasure of the old value of* k_w. Thus a client that wishes to delete an ensemble key k_w permanently at the end of its lifecycle can do so with a Reset call.

Reset is an authenticated call, and thus requires the following capability.

Authentication. To authenticate itself for API calls, the client must first invoke GetAuth, which has the server transmit an (encrypted) authentication token *authtoken* to the client out-of-band. The token expires after a period of time determined by a configuration parameter in PYTHIA. Our current implementation uses e-mail for this, see Section 3.5 for more details. Of course, in some deployments one may want authentication to be performed in other ways, such as tokens dispensed by administrators (for enterprise settings) or simply given out on a first-come-first-serve basis for each ensemble identifier (for public Internet services).

3.4.1 Implementation

We implemented a prototype PYTHIA PRF service as a web application accessed over HTTPS. All requests are first handled by an nginx web server with uWsgi as the application server gateway that relays requests to a Django back-end. The PRF-Srv functionality is implemented as a Django module written in Python. Storage for the server's key table and rate-limiting information is done in MongoDB.

We use the Relic cryptographic library [15] (written in C) with our own Python wrapper. We use Barreto-Naehrig 254-bit prime order curves (BN-254) [23]. These curves provide roughly 128-bits of security.

In our experiments the service is run on a single (virtual) machine, but our software stack permits components (web server, application sever, database) to be distributed among multiple machines with updates to configuration files.

For the purpose of comparison, we implemented three variants of the Pythia service. The first two are the unblinded protocols described in Section 3.3. In these two schemes, the client sends m in the clear (possibly hashed with a secret salt value first) and the server replies with $y = H_1(t\|m)^k$. In the first scheme, denoted UNB, the server provides $p = g_1^k$ and a zero-knowledge proof where g_1 is a generator of $\mathbb{G}_1$. The second scheme, denoted BLS, uses a BLS signature for verification. The server provides $p = g_2^k$ where g_2 is a generator of $\mathbb{G}_2$ and the client verifies the response by computing and comparing the values: $e(y, g_2) = e(H_1(t\|m), p)$.

Our partially-oblivious scheme is denoted PO.

For the evaluation below we use a Python client implementing PRF-Cl for all three schemes using the same libraries indicated above for the server and httplib2 to perform HTTPS requests.

3.4.2 Performance

For performance and scalability evaluation we hosted our Pythia server implementation on Amazon's Elastic Compute Cloud (EC2) using a c4.xlarge instance which provides 8 virtual CPUs (Intel Xeon third generation, 2.9GHz), 15 GB of main memory, and solid state storage. The web server, nginx, was configured with basic settings recommended for production deployment including one worker process per CPU.

Latency. We measured client query latency for each protocol using two clients: one within the same Amazon Web Service (AWS) availability zone (also c4.xlarge) and one hosted at the University of Wisconsin–Madison with an Intel Core i7 CPU (3.4 GHz). We refer to the first as the LAN (local-area network) setting and the second as the WAN (wide-area network) setting. In the LAN settings we used the AWS internal IP address. All queries were made over TLS and measurements include the time required for clients to blind messages and unblind results (PO), as well as

Group	Time (μs) Group Op	Exp	Hashing
$\mathbb{G}_1$	5.7	175	77
$\mathbb{G}_2$	6.7	572	210
$\mathbb{G}_T$	9.8	1145	–
pairing operation (e) takes 1005 μs			

Table 3.2: Time taken by each operation in BN-254 groups. Hashing times are for 64-byte inputs.

Server Op	Time (ms)		
Table	1.2		
Rate-limit	0.9		
	UNB	**BLS**	**PO**
Sign	0.3	0.3	1.5
Prove	0.5	0.3	2.5

Client Op	UNB	BLS	PO
Blind	-	-	0.3
Unblind	-	-	1.2
Verify	0.9	2.0	4.0

Table 3.3: Computation time for major operations to perform a PRF evaluation. Table retrieves $K[w]$ from database; Rate-limit updates rate-limiting record in database; and Sign generates the PRF output;

verify proofs provided by the server (unless indicated otherwise). All machines used for evaluation were running Ubuntu 14.04.

Microbenchmarks for group operations appear in Table 3.4.2 and Table 3.3 shows the timing of individual operations that comprise a single PRF evaluation. All results are mean values computed over 10,000 operations. These values were captured on an EC2 c4.xlarge instance using the Python profiling library line_profiler. The most expensive operations, by a large margin, are exponentiation in $\mathbb{G}_t$ and the pairing operation. By extension, PO sign, prove, and verify operations become expensive.

We measured latencies averaged over 1,000 PRF requests (with 100 warmup requests) for each scheme and the results appear in Table 3.4. Computation time dominates in the LAN setting due to the almost negligible network latency. The WAN case with cold connections (no HTTP KeepAlive) pays a performance penalty due to the four round-trips required to set up a new TCP and TLS connection. While even 400 ms latencies are not prohibitive in our applications, straightforward engineering improvements would vastly improve WAN timing: using TLS session

	Latency (ms)					
	LAN			**WAN**		
Scheme	Cold	Hot	No π	Cold	Hot	No π
UNB	7.0	3.8	2.4	389	82	80
BLS	7.9	4.9	2.4	392	85	80
PO	14.9	11.8	5.2	403	96	84
RTT ping	0.1			82		

Table 3.4: Average latency to complete a PRF-Cl with client-server communication over HTTPS. LAN: client and server in the same EC2 availability zone. WAN: server in EC2 US-West (California) and client in Madison, WI. Hot connections made with HTTP KeepAlive enabled; cold connections with KeepAlive disabled. No π: KeepAlive enabled; prove and verify computations are skipped.

resumption, using lower-latency secure protocol like QUIC [172], or even switching to a custom UDP protocol (for an example one for oblivious PRFs, see [29]).

Throughput. We used the distributed load testing tool autobench to measure maximum throughput for each scheme. We compare to a static page containing a typical PRF response served over HTTPS as a baseline. We used two clients in the same AWS region as the server. All connections were cold: no TLS session resumption or HTTP KeepAlive. Results appear in Figure 3.4. The maximum throughput for a static page is 2,200 connections per second (cps); UNB and BLS 1,400 cps; and PO 1,350 cps. Thus our Pythia implementation can handle a large number of clients on a single EC2 instance. If needed, the implementation can be scaled with standard techniques (e.g., a larger number of web servers and application servers on the front-end with a distributed key-value store on the back-end).

Storage. Our implementation stores all ensemble pre-key table (K) entries and rate-limiting information in MongoDB. A table entry is two 32 byte values: a SHA-256 hash of the ensemble selector w and its associated value $\mathtt{K}[w]$. In MongoDB the average storage size is 195 bytes per entry (measured as the average of 100K entries), including database overheads and indexes. This implementation scales easily to 100 M clients with under 20 GB of storage.

To rate-limit queries, our implementation stores tweak values along with a counter and a timestamp (to expire old entries) in MongoDB. Tweak values are also hashed using SHA-256 which ensures entries are of constant length. In our implementation each distinct tweak requires an average of 144 bytes per entry (including overheads

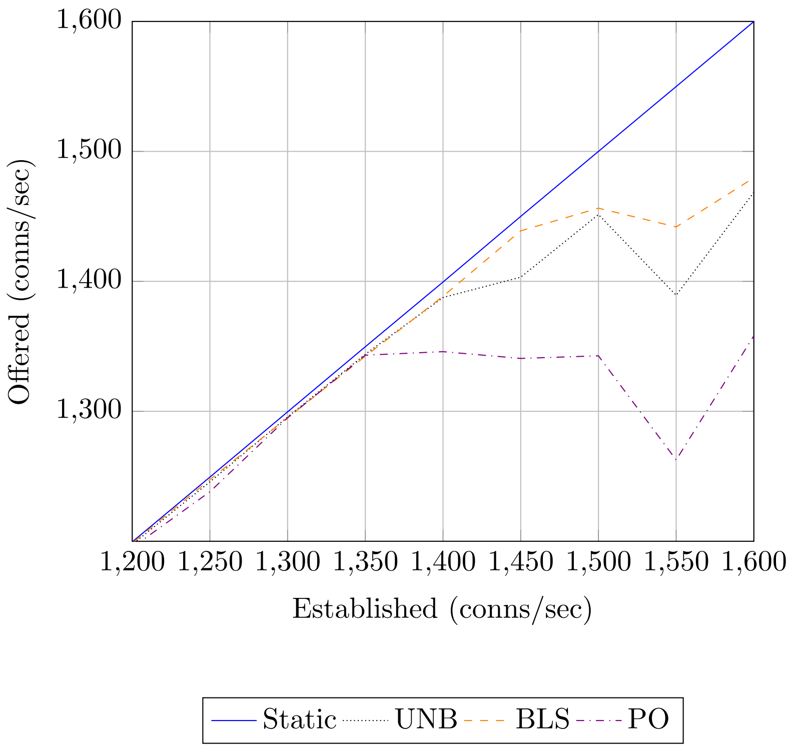

Figure 3.4: Throughput of PRF-Srv requests and a static page request over HTTPS measured using two clients and a server hosted in the same EC2 availability zone.

and indexes). Note however that rate limiting entries are purged periodically as counts are only required for one rate-limiting period. Our implementation imposes rate-limits at hour granularity. Assuming a maximum throughput of 2,000 requests per second, rate-limiting storage never exceeds 1 GB.

All told, with only 20 GB stored data, PYTHIA can serve over 100 M clients and perform rate-limiting at hour granularity. Thus fielding a database for PYTHIA can be accomplished on commodity hardware.

3.5 Additional Pythia API Details

Many PYTHIA-dependent services can benefit from additional API features and calls beyond the primary ones discussed previously. (For example, the PYTHIA password onion system in Section 3.7 uses the Transfer API call.) We detail these other API features in this section.

Key-management options. The client can specify a number of options in the call Init regarding management of the ensemble key k_w. The client can provide a contact email address to which alerts and authentication tokens may be sent. (If no e-mail is given, no API calls requiring authentication are permitted at present and no alerts are provided. Later versions of PYTHIA will support other authentication and alerting methods.)

The client can specify whether k_w should be resettable (default is “yes”). The client can specify a limit on the total number of F_{k_w} queries that should be allowed before resetting $\mathtt{K}[w]$ (default is unlimited) and/or an absolute expiration date and time in UTC at which point $\mathtt{K}[w]$ is deleted (default is no time-out). Either of these options overrides the resettable flag. The client can specify a public key pk_{cl} for a public-key encryption scheme under which to encrypt authentication tokens and update tokens (for Reset, Transfer, as described below, and for master secret key rotations). Finally, the client can request that alerts be sent to the contact email address in the case of rate limit violations. This option is ignored if no contact email is provided. The options are summarised in Table 3.5.

Selector option	Description
Email	Contact email for selector
Resettable	Whether client-requested rotations allowed
Limit	Establish rate-limit per t
Time-out	Date/time to delete k_w
Public-key	Key under which to encrypt and store update and authentication tokens
Alerts	Whether to email contact upon rate limit violation

Table 3.5: Optional settings for establishing key selectors in PYTHIA.

Command	Description
$\mathsf{Transfer}(w, w' \, [, options])$	Creates new ensemble w'; outputs update token $\Delta_{w \to w'}$; resets k_w
$\mathsf{SendTokens}(w, authtoken)$	Sends stored update tokens to client
$\mathsf{PurgeTokens}(w, authtoken)$	Purges all stored update tokens for ensemble w

Table 3.6: The PYTHIA API. The individual calls are explained in detail in the text.

PYTHIA also offers some additional API calls, given in Table 3.6, which we now describe.

Ensemble transfer. A client can create a new ensemble w' (with the same options as in Init) while receiving an update token that allows PRF outputs under ensemble w to be rolled forward to w'. This is useful for importing a password database to a new server. The PYTHIA service returns an update token $\Delta_{w \to w'}$ for this purpose and stores it encrypted under pk_{cl}. For the case $w' = w$, this call also allows optional updates on an existing ensemble w.

Update-token handling. The PYTHIA service stores update tokens encrypted under pk_{cl}, with accompanying timestamps for versioning. The API call SendTokens causes these to be e-mailed to the client, while PurgeTokens causes update-token ciphertexts to be deleted from PYTHIA.

Note that once an update token is deleted, old PRF values to which the token was not applied become cryptographically erased — they become random values unrelated to any messages. A client can therefore delete the key associated with an ensemble by calling Reset and PurgeTokens.

Master secret rotations. PYTHIA can also rotate its master secret key msk to a new key msk'. Recall that ensemble keys are computed as $k_w = \mathrm{HMAC}(msk, \mathtt{K}[w])$, so rotation of msk results in rotation of all ensemble keys. To rotate to a new msk', the server computes k_w for all ensembles w with entries in $\mathtt{K}$, and stores δ_w encrypted under pk_{cl}. If no encryption key is set, then the token is stored in the clear. This is a forward-security issue while it remains, but only for that particular key ensemble. At this point msk is safe to delete. Clients can be informed of the key rotation via e-mail.

Subsequent SendTokens requests will return the resulting update token, along with any other stored update tokens for the ensemble. If multiple rotations occur between client requests, then these can be aggregated in the stored update token for each ensemble. This is trivial if they are stored in the clear (just multiply the new token against the old) and also works if they are encrypted with an appropriately homomorphic encryption scheme such as ElGamal [101].

3.6 Formal Security Analyses

We provide formal security notions for partially oblivious PRFs, and proofs of security relative to them for our scheme from Section 3.3. We note that we do not cover all of the proposed features. In particular, we do not cover verifiability, nor the security of key rotations as part of our security analysis.

A more comprehensive treatment of partially-oblivious functions in the context of password hardening can be found in the follow-up work by Lai et al. [125], which we discuss further in Section 3.9.

Partially-oblivious PRFs. A partially oblivious PRF protocol $\Pi = (\mathsf{KeyGen}, \text{PRF-Cl}, \text{PRF-Srv}, F)$ consists of the following. The key generation algorithm KeyGen outputs a public key and private key pair (pk, sk). We assume that from sk one can compute pk easily. The PRF-Srv algorithm takes input the secret key sk and a client request message (a bit string) and returns a server response message (another bit string). The client algorithm PRF-Cl takes inputs a tweak t and message m, can make a single call to PRF-Srv, and outputs a value. Finally we associate to the protocol a keyed function $F_{sk} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. A scheme is correct if executing $\text{PRF-Cl}^{\text{PRF-Srv}_{sk}(\cdot)}(t, m)$ with fresh coins matches $F_{sk}(t, m)$ with probability one. In words, the protocol computes the appropriate function of t, m.

Bilinear pairing setups. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups all of order p that have associated to them an admissible bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Recall that for generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, there exists a generator $g_T \in \mathbb{G}_T$ such that $e(g_1^\alpha, g_2^\beta) = g_T^{\alpha\beta}$ for all $\alpha, \beta \in \mathbb{Z}_p$. As shorthand for below we refer to a pairing setup $\mathbb{G} = (g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and assume some compact description of $\mathbb{G}$ as a bit-string where appropriate.

The scheme. The partially-oblivious PRF at the core[2] of our bilinear pairing scheme from Section 3.3 is as follows for some fixed pairing setup $\mathbb{G}$. Let $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \{0,1\}^* \to \mathbb{G}_2$ be hash functions that we will later model as random oracles.

Key generation KeyGen picks a random exponent sk and computes a public key $pk = g_1^{sk}$. The PRF-Cl(t, m) algorithm computes a mask $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and sends t and $x = H_2(m)^r$ to the server. The PRF-Srv(sk, t, x) computes $y = e(H_1(t), x)^{sk}$ and a ZKP π that $\text{DL}_{g_1}(pk) = \text{DL}_{\tilde{x}}(y)$ where $\tilde{x} = e(H_1(t), x)$. It sends pk, y, π to the client, who verifies the ZKP, deletes it, and then outputs $y^{1/r}$. The correctness of the scheme follows from the correctness of the ZKP and the properties of the pairing.

The ZKP is used to ensure that a malicious server responds as per the protocol. In the following security analyses we focus primarily on malicious clients, and for simplicity analyse a simpler version of the protocol that omits the ZKP. The proofs found below can be extended to the full protocol by applying the zero-knowledge security of the proof systems that we use (i.e., use the zero-knowledge simulator to produce fake, but realistic-looking to the client, proofs).

[2]For brevity we omit key selectors here, and instead focus on analysing security for a single key instance. Assuming properly generated keys for each selector, one can show that security for a single key instance implies security for many.

```
Game om-UNP$_\Pi^\mathcal{A}$
$(pk, sk) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$c \leftarrow 0$
$(t_1, m_1, \sigma_1), \ldots, (t_\ell, m_\ell, \sigma_\ell) \leftarrow\!\!\$\ \mathcal{A}^{\text{PRF-Srv}}$
if $\exists i \neq j\ .\ (t_i, m_i) = (t_j, m_j)$ then
    return false
return $(\wedge_i(\sigma_i = F_{sk}(t_i, m_i)) \wedge c < \ell)$

PRF-Srv$(t, Y)$
$c \leftarrow c + 1$
return PRF-Srv$_{sk}(t, Y)$
```

Figure 3.5: Security game for one-more unpredictability.

3.6.1 Unpredictability Security

We define a one-more unpredictability security notion. It modifies one-more unforgeability [160] to be suitable for the setting of unpredictable functions (as opposed to publicly verifiable signatures). The game is shown in Figure 3.5. We associate to any protocol Π, adversary $\mathcal{A}$, and query number q the one-more-unpredictability advantage defined as

$$\mathsf{Adv}_{\Pi,q}^{\text{om-unp}}(\mathcal{A}) = \Pr\left[\text{om-UNP}_{\Pi,q}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] .$$

The probability here (and for games defined later below) is over all random coins used by the procedures and the adversary. The event refers to the probability that the value returned by the main procedure is true. In words, the definition requires that an adversary cannot produce ℓ outputs of the PRF using less than ℓ queries on partially-blinded inputs to the server. One can easily extend this notion to deal with full blinded inputs as well, but we will not need this.

This notion of security is sufficient for PYTHIA in applications where the output of the protocol is not stored, but rather used as an unforgeable credential such as with our hardened Brainwallet application (Section 3.8).

The security of our scheme is based on the following one-more bilinear computational Diffie-Hellman (BCDH) problem, an extension of the one-more CDH assumption given by Boldyreva [56]. To the best of our knowledge this assumption is new, but

```
Game om-CDH$_{\mathbb{G}}^{\mathcal{B}}$
$sk \leftarrow_{\$} \mathbb{Z}_p$
$q_h, q_{1,t}, q_{2,t} \leftarrow 0$
$(i_1, j_1, \sigma_1), \ldots, (i_\ell, j_\ell, \sigma_\ell) \leftarrow_{\$} \mathcal{A}^{\mathrm{Targ}_1, \mathrm{Targ}_2, \mathrm{Help}}(\mathbb{G}, g_1^{sk})$
if $q_h \geq \ell$ then
  return false
if $\exists \alpha \,.\, (i_\alpha > q_{1,t}) \vee (j_\alpha > q_{2,t})$ then
  return false
if $\exists \alpha \neq \beta \,.\, (i_\alpha, j_\alpha) = (i_\beta, j_\beta)$
  return false
return $\forall \alpha \,.\, e(X_{i_\alpha}, Y_{j_\alpha})^{sk} = \sigma_\alpha$
```

```
Targ$_1$
$q_{1,t} \leftarrow q_{1,t} + 1$
$X_{q_{1,t}} \leftarrow_{\$} \mathbb{G}_1$
return $X_{q_{1,t}}$

Targ$_2$
$q_{2,t} \leftarrow q_{2,t} + 1$
$Y_{q_{2,t}} \leftarrow_{\$} \mathbb{G}_2$
return $Y_{q_{2,t}}$

Help($Z$)
$q_h \leftarrow q_h + 1$
return $Z^{sk}$
```

Figure 3.6: Security game for a one-more BCDH assumption for bilinear pairing setting $\mathbb{G} = (g_1, g_2, g_t, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$.

it is a straightforward adaptation of previous one-more assumptions [32, 56] to our setting. For a pairing setup $\mathbb{G}$, game om-CDH$_{\mathbb{G}}$ is defined in Figure 3.6. In words, the adversary gets a group element $g_1^{sk} \in \mathbb{G}_1$ as well as target oracles Targ$_1$, Targ$_2$ that return random group elements in $\mathbb{G}_1, \mathbb{G}_2$ respectively. Finally the adversary can query a helper oracle Help that raises $\mathbb{G}_T$ elements to the k. To win, it must compute ℓ values $e(X_i, Y_j)^{sk}$ for ℓ larger than the number of helper queries and each X_i, Y_j a unique pair of (distinct) values returned by the target oracle. Let $\mathsf{Adv}_{\mathbb{G}}^{\text{om-bcdh}}(\mathcal{B}) = \Pr\left[\text{om-CDH}_{\mathbb{G}}^{\mathcal{B}} \Rightarrow \mathsf{true}\right]$.

We have the following theorem establishing the one-more unpredictability of our scheme The proof is essentially identical to the proof of Boldyreva's blind signatures [56].

Theorem 3. *Let* Π *be the simplified partially oblivious PRF protocol for a pairing setup* $\mathbb{G}$ *and* H_1, H_2 *modelled as random oracles. Then for any one-more unpredictability adversary* $\mathcal{A}$ *making at most* q PRF-Srv *queries, we give in the proof below a one-more CDH adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{om-unp}}_{\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\text{om-bcdh}}_{\mathbb{G}}(\mathcal{B}),$$

where $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus $\mathcal{O}(q)$ group operations.

Proof. We assume without loss of generality that $\mathcal{A}$ never repeats a query to either random oracle and makes a random oracle $H_1(t_i)$ and $H_2(m_i)$ query for each (t_i, m_i, σ_i) triple it outputs. The adversary $\mathcal{B}$ will work as follows when given inputs $\mathbb{G}, X$ and access to oracles $\mathrm{Targ}_1, \mathrm{Targ}_2, \mathrm{Help}$. First, it runs $\mathcal{A}$. Whenever $\mathcal{A}$ makes an $H_1(t)$ query, $\mathcal{B}$ queries Targ_1 to obtain a $\mathbb{G}_1$-element that we will denote $X[t]$, sets c_t to be the number of H_1 queries so far (including the current), and returns $X[t]$ to $\mathcal{A}$. Whenever $\mathcal{A}$ makes an $H_2(m)$ server query, $\mathcal{B}$ queries Targ_2, obtains a $\mathbb{G}_2$-element that we will denote $Y[m]$, sets d_m to be the number of H_2 queries so far (including the current), and returns $Y[m]$ to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a PRF-Srv(t, Y) query, the adversary $\mathcal{B}$ computes $Z \leftarrow e(H_1(t), Y)$, and then queries Z to its helper oracle Help to obtain a value $\sigma \in \mathbb{G}_T$. It returns σ to $\mathcal{A}$.

Eventually $\mathcal{A}$ outputs a series of triples $(t_1, m_1, \sigma_1), \ldots, (t_q, m_q, \sigma_q)$. At this point adversary $\mathcal{B}$ outputs the sequence of pairs $(c_{t_1}, d_{m_1}, \sigma_1), \ldots, (c_{m_q}, d_{m_q}, \sigma_q)$.

Suppose $\mathcal{A}$ wins its game. Then it made at most $q-1$ queries to PRF-Srv and so $\mathcal{B}$ makes at most $q-1$ queries to Help. It is also the case that all predictions by $\mathcal{A}$ are for unique tag, message pairs, meaning that $\mathcal{B}$'s output will also be for unique pairs of targets. Finally, it is clear that correct predictions σ_i are also BCDH solutions. □

3.6.2 Pseudorandomness Security

Unpredictability security, like unforgeability, is not sufficient in all applications. In particular, it could be that a protocol produces unforgeable outputs but each output leaks everything about the message. So in our use of Pythia for storage of hardened password hashes we need something more. Ideally we would prove a version of oblivious PRF security suitably adapted for the partially oblivious setting. However, we do not believe our schemes can be proven secure relative to such notions since they require "programming" the PRF outputs. One might adapt our schemes to meet them, but the most efficient adaptation — considering instead $\tilde{F}_k(t, m) = H(t \| m \| e(H_1(t), H_2(m))^k)$ as in [117] — prevents one from performing key updates. At the same time, we could find no attacks that exploit the algebraic structure revealed by storing the unhashed output. This leaves the question of what level of security can be proven about our scheme.

We introduce a new notion called one-more PRF security. Intuitively for a scheme that meets it, an attacker that interacts $q-1$ times with PRF-Srv still cannot distinguish one more evaluation of F_k from a random point. This appears to capture the security properties we require in the web server compromise case: even if the adversary breaks in, it can only distinguish from random points as much of the stored hardened hashes as queries to PRF-Srv.

To build up some intuition towards a formal notion, consider giving an adversary two oracles. The first is a real-or-random function oracle RoR and the second is an oracle for the server's implementation of the protocol PRF-Srv. In a normal PRF game one would simply have RoR reply either always with $F_k(t, m)$ upon query t, m or always with a fresh random point. (Assume the adversary never repeats a query to RoR.) But this game is trivial to win because the adversary can simply use the PRF-Srv oracle to compute $F_k(t, m)$ and check if it matches the value returned by RoR(t, m). We might want to somehow restrict queries to PRF-Srv but there seems no way to do this given the ability of clients to blind their messages.

Instead we take a different route, adapting the concept of one-more unpredictability to a pseudorandomness setting. Let Π be a partially-oblivious PRF protocol. The game om-PRF$_\Pi$ is defined in Figure 3.7. It gives the adversary a challenge oracle RoR to which it can query (t, m) pairs. The oracle flips a fresh challenge bit and responds accordingly. We restrict attention to adversaries that never repeat a query to RoR. Finally the adversary gets access to a PRF-Srv oracle. The adversary's task is to determine all of the challenge bits, and we measure this by asking that it guess the XOR of them. This XOR measure is borrowed from the multi-instance security notions[3] of Bellare, Ristenpart, and Tessaro [33]. We associate to an adversary $\mathcal{A}$ and Π the advantage measure defined by

$$\mathsf{Adv}_{\Pi}^{\text{om-prf}}(\mathcal{A}) = \left|2 \cdot \Pr\left[\text{om-PRF}_{\Pi}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\right| \ .$$

[3]Our one-more notions are *not* measuring multi-instance security in the sense of [33] as the same key underlies all challenges.

```
Game om-PRF$^{\mathcal{A}}_{\Pi,\nu}$
$(pk, k) \leftarrow_\$ \mathsf{KeyGen}$
$q, c \leftarrow 0$
$(i_1, \ldots, i_\ell, b') \leftarrow_\$ \mathcal{A}^{\mathrm{RoR},\mathrm{PRF\text{-}Srv}}$
if $\ell > q$ OR $c \geq \ell$ then
    return $\mathsf{false}$
if $\exists \alpha \neq \beta \,.\, i_\alpha = i_\beta$ then
    return $\mathsf{false}$
return $b' = \bigoplus_{\alpha=1}^{\ell} \vec{b}[i_\alpha]$
```

```
RoR$(t, m)$
$q \leftarrow q + 1$
$\vec{b}[q] \leftarrow_\$ \{0, 1\}$
$Z_1 \leftarrow F_k(t, m)$
$Z_0 \leftarrow_\$ \mathsf{Rng}$
return $Z_{\vec{b}[i]}$
```

```
PRF-Srv$(t, Y)$
$c \leftarrow c + 1$
return PRF-Srv$_k(t, Y)$
```

Figure 3.7: Security game for one-more pseudorandomness.

Note that for any correct protocol Π there exists an efficient adversary that can win the game with probability $1/2$ by querying RoR once on $(1, t, m)$ for arbitrary t, m and outputting $(1, b')$ for randomly chosen b'. Hence the advantage measure is scaled as shown. The use of XOR ensures that even if one can solve $q - 1$ challenges (e.g., using the PRF-Srv oracle) determining b' for q challenges requires gaining some advantage over the remaining challenge bit.

Relationship with PRF security. This one-more PRF notion is a strict strengthening of the conventional PRF security. Consider the following formulation of PRF security due originally to Goldreich, Goldwasser and Micali [111]. An adversary $\mathcal{A}$ is given access to two oracles, one that returns $F_k(t, m)$ on query t, m of their choosing, and one that upon query t^*, m^* of the adversary's choosing flips a bit b and returns either $F_k(t^*, m^*)$ or a random point according to the bit. We allow $\mathcal{A}$ to make multiple queries to the first oracle but only a single to the second.

We now sketch a proof showing that PRF security as defined above is implied by one-more PRF security. Consider any such PRF adversary $\mathcal{A}$ against $F_k(t, m)$. We can build a one-more PRF adversary $\mathcal{B}$ whose advantage upper bounds $\mathcal{A}$'s PRF advantage, as follows. To any $F_k(t, m)$ query by $\mathcal{A}$, the adversary $\mathcal{B}$ first queries t, m to its own RoR oracle and also runs PRF-Cl(t, m) using its PRF-Srv oracle in order to compute $F_k(t, m)$. It determines the challenge bit for this RoR query and returns $F_k(t, m)$. When $\mathcal{A}$ makes a query to its second oracle, the challenge oracle, adversary $\mathcal{B}$ queries its RoR oracle and returns the result. When $\mathcal{A}$ outputs a bit b', the adversary $\mathcal{B}$ outputs b' XOR'd with each of the already-solved challenge bits.

It is easy to analyse this formally and show that $\mathcal{B}$ wins whenever $\mathcal{A}$ would in the PRF game.

Relationship with unpredictability. Showing that one-more PRF security is strictly stronger than one-more unpredictability is straightforward. Consider an adversary $\mathcal{A}$ who computes ℓ tuples (t_i, m_i, σ_i), where $\sigma_i = F(t_i, m_i)$, and $c < \ell$ queries to PRF-Srv. An adversary $\mathcal{B}$ can use $\mathcal{A}$ to win the om-PRF game with the same advantage by forwarding queries to the corresponding oracle, and wins the game by setting b_i based on whether $\mathrm{RoR}(t_i, m_i) = \sigma_i$.

However, given a one-more unpredictable function F, the function $F'(t, m) := F(t, m) \| m$ is still unpredictable, but can be trivially distinguished.

Analysing our scheme. We now analyse the security of our partially-oblivious PRF scheme from Section 3.3. To do so we introduce a new hardness assumption that is a generalisation of the bilinear Decisional Diffie-Hellman (BDDH) assumption underlying the conventional PRF security of $F_k(t, m) = e(H_1(t), H_2(m))^k$. (The latter is a corollary of a result due to Boneh and Waters [63].) The new assumption is analogous to the one-more BCDH assumption given above.

Fix some pairing setting $\mathbb{G} = (g_1, g_2, g_t, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and refer to the game shown in Figure 3.8. The game tasks the adversary $\mathcal{B}$ in distinguishing one more BDDH instance than the number of helper queries it makes. That is, the adversary is given a value g_1^k and target oracles $\mathrm{Targ}_1, \mathrm{Targ}_2$ which give back random points in $\mathbb{G}_1$ and $\mathbb{G}_2$. The adversary can query a challenge oracle $\mathrm{Chal}(i, j)$, which gives back either $e(X, Y)^k$ for X, Y each being previously returned by the respective target oracles (if the challenge bit is one) or gives back a random point $Z \in \mathbb{G}_T$ (if the challenge bit is zero). Later we will often allow an adversary to query $\mathrm{Chal}(X, Y)$, with the implied meaning hopefully obvious. The helper oracle returns Z^k for any queried Z, effectively allowing the attacker to trivially solve a single BDDH instance. We define advantage of an adversary $\mathcal{B}$ against a setup $\mathbb{G}$ by

$$\mathsf{Adv}_{\mathbb{G}}^{\text{om-cdh}}(\mathcal{B}) = \left|2 \cdot \Pr\left[\text{om-CDH}_{\mathbb{G}}^{\mathcal{B}} \Rightarrow \mathsf{true}\right] - 1\right|$$

```
Game om-CDH$^{\mathcal{B}}_{\mathbb{G}}$
$k \leftarrow_\$ \mathbb{Z}_p$
$q_{ch}, q_{1,t}, q_{2,t}, c \leftarrow 0$
$(i_1, \ldots, i_\ell, b') \leftarrow_\$ \mathcal{B}^{\mathrm{Chal},\mathrm{Targ}_1,\mathrm{Targ}_2,\mathrm{Help}}(\mathbb{G}, g_1^k)$
if $\ell > q_{ch} \mathrm{OR} c \geq \ell$ then
    return false
if $\exists \alpha \neq \beta \,.\, i_\alpha = i_\beta$ then
    return false
return $b' = \bigoplus_{\alpha=1}^{\ell} \vec{b}[i_\alpha]$
```

```
Chal$(i, j)$
if $q_{1,t} < i \mathrm{OR} q_{2,t} < j$ then
    return $\perp$
$q_{ch} \leftarrow q_{ch} + 1$
$\vec{b}[q_{ch}] \leftarrow_\$ \{0, 1\}$
$Z_1 \leftarrow e(g_1, g_2)^{\gamma_1[i] \cdot \gamma_2[j] \cdot k}$
$Z_0 \leftarrow_\$ \mathbb{G}_T$
return $Z_{q_{ch}}$
```

```
Targ$_1$
$q_{1,t} \leftarrow q_{1,t} + 1$
$\gamma_1[q_{1,t}] \leftarrow_\$ \mathbb{Z}_p$
return $g_1^{\gamma_1[q_{1,t}]}$
```

```
Targ$_2$
$q_{2,t} \leftarrow q_{2,t} + 1$
$\gamma_2[q_{2,t}] \leftarrow_\$ \mathbb{Z}_p$
return $g_2^{\gamma_2[q_{2,t}]}$
```

```
Help$(Z)$
$c \leftarrow c + 1$
return $Z^k$
```

Figure 3.8: Game for the one-more BDDH assumption for bilinear pairing setting $\mathbb{G} = (g_1, g_2, g_t, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$.

Note that if $\mathcal{B}$ makes two Targ queries, makes a single Chal$(1, 1)$, and never uses Help, then this is exactly the classic BDDH assumption. Thus this assumption is stronger than BDDH. Results from [65] and [67] suggest that one-more problems of this type are possibly easier, but the verdict is still out.

We have the following theorem.

Theorem 4. *Let* Π *be the partially-oblivious PRF scheme for pairing setup* $\mathbb{G}$ *and with* H_1, H_2 *modelled as random oracles. Let adversary* $\mathcal{A}$ *be a one-more PRF adversary making* q *queries to* $\mathrm{RoR}(\cdot, \cdot)$, h_1 *queries to* H_1, h_2 *queries to* H_2, *and* $c < q$ *queries to* PRF-Srv. *Then we give in the proof below a one-more BDDH adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{om-prf}}_{\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\text{om-cdh}}_{\mathbb{G}}(\mathcal{B}) \ .$$

Adversary $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus $\mathcal{O}(h_1 + h_2 + c)$ group or pairing operations, makes at most q challenge queries, c helper queries, h_1 queries to Targ_1 *and h_2 queries to* Targ_2*.*

Proof. This proof follows much the same steps as the one-more unpredictable proof given previously. As before, we assume without loss of generality that $\mathcal{A}$ makes all queries to H_1 and H_2 at the start, and never makes redundant queries. We construct an adversary $\mathcal{B}$ for the om-PRF game as shown in Figure 3.9.

Adversary $\mathcal{B}$ responds to any query to $\mathrm{HSim}_1(t)$ with a point $X[t] \in \mathbb{G}_1$, where $X[t]$ is sampled from Targ_1. Similarly, for queries to $\mathrm{HSim}_2(m)$, $\mathcal{B}$ returns a point $Y[m]$ sampled from Targ_2.

Whenever $\mathcal{A}$ queries $\mathrm{RoRSim}(t, m)$, $\mathcal{B}$ calls the challenge oracle $\mathrm{Chal}(X[t], Y[m])$, and returns the response to $\mathcal{A}$. Notice that this challenge precisely matches the expected real or random challenge.

Furthermore, on queries (t, Y) to SrvSim, $\mathcal{B}$ computes $e(H_1(t), Y)$, and submits this value to the Help oracle.

In this way, $\mathcal{B}$ accurately simulates all queries in the om-PRF game.

Finally, suppose $\mathcal{A}$ outputs a tuple $(i_1, i_2, \ldots, i_\ell)$ and a bit b'. Then $\mathcal{B}$ outputs the same tuple, and bit. If $\mathcal{A}$ wins its game, then it has distinguished ℓ real or random instances of the PRF, with $c < \ell$ queries to the PRF-Srv oracle. It is easy to see that $\mathcal{B}$ has also distinguished ℓ challenges with $c < \ell$ queries to the Help oracle.

$\square$

3.6.3 Relationship with Fully Oblivious PRFs

Jarecki et al. [117] propose a universally composable model of verifiable, oblivious pseudorandom functions. This model is close to the functionality required for our PRF service. However, these constructions do not support key updates because hash functions are applied to several values in the course of the protocol. Hashing destroys any algebraic structure that could be exploited to enable key rotation.

```
adversary $\mathcal{B}^{\text{Chal},\text{Targ},\text{Help}}(\mathbb{G}, K)$
    $pk \leftarrow e(g_1, K)$
    $(i_1, \ldots, i_\ell, b') \leftarrow_\$ \mathcal{A}^{\text{RoRSim},\text{SrvSim},\text{HSim}_1,\text{HSim}_2}(pk)$
    return $(i_1, \ldots, i_\ell, b')$

RoRSim$(t, m)$
    return Chal$(X[t], Y[m])$

HSim$_1(t)$                      HSim$_2(m)$
    $X[t] \leftarrow \text{Targ}_1()$        $Y[m] \leftarrow \text{Targ}_2()$
    return $X[t]$                    return $Y[m]$

SrvSim$(t, Y)$
    $c \leftarrow c + 1$
    return Help$(e(X[t], Y))$
```

Figure 3.9: Adversary $\mathcal{B}$ used in proof of Theorem 4.

We informally summarise the ideal functionality $\mathcal{F}_{\textsf{v-OPRF}}$ of the Jarecki et al. model as follows:

1. First, $\mathcal{F}_{\textsf{v-OPRF}}$ performs the key-generation and parameter registering functions (for example, distributing public keys).

2. A user $\mathcal{U}$ submits a message x and a server $\mathcal{S}$ to evaluate the message. Communication is initiated with the server S.

3. The oblivious PRF protocol takes place between $\mathcal{U}$ and $\mathcal{S}$. This is equivalent to the PRF-Srv protocol.

4. When both $\mathcal{U}$ and $\mathcal{S}$ have finished communicating, a flag is generated which denotes whether the verifiable property is satisfied.

5. $\mathcal{U}$ learns the value y which is set to be the output of the PRF on x.

Moreover, the following must hold:

- The output y should be indistinguishable from a randomly chosen value.

- The server $\mathcal{S}$ evaluates inputs on at most as many points which are received by $\mathcal{U}$.

This is a simplified version of the functionality from [117], where it is described in the universal composability framework. Under the UC framework, adversarial capabilities include statically corrupting both users and servers, and complete control over the communication channel.

The security is measured by the probability that an environment can distinguish whether it is interacting with the real world (as defined above), or a simulated version of the ideal world.

Using this simplified version, we prove the following.

Proposition 1. *Let $\mathcal{F}_{\textsf{v-OPRF}}$ be a verifiable, oblivious PRF as described above, and let Π be the partially oblivious PRF derived from $\mathcal{F}_{\textsf{v-OPRF}}$. Then for any adversary $\mathcal{A}$ against the one-more pseudorandomness of Π we construct an adversary $\mathcal{B}$ such that*

$$\left| \mathsf{Adv}^{\text{om-prf}}_{\Pi,q}(\mathcal{A}) - \frac{1}{2} \right| \leq \mathsf{Adv}^{\text{voprf}}_{\mathcal{F}_{\textsf{v-OPRF}}}(\mathcal{B}).$$

Proof. (Sketch) For every query to the PRF-Srv oracle from $\mathcal{A}$, $\mathcal{B}$ simulates the same functionality in the v-OPRF game. For the RoR queries, the v-OPRF adversary chooses to return either the actual evaluation of the query, or a random value with probability 1/2.

If $\mathcal{B}$ is interacting with the ideal world, then all RoR queries will be random, even though half of the queries are ‘real’. In this case, $\mathcal{A}$ cannot win with probability better than 1/2.

On the other hand, if $\mathcal{B}$ is interacting with the simulation, then half of the RoR queries are true evaluations of the function $\mathcal{F}_{\textsf{v-OPRF}}$. Therefore, this is an accurate simulation of the om-PRF game, and $\mathcal{A}$ wins with probability $\mathsf{Adv}^{\text{om-prf}}_{\Pi,q}(\mathcal{A})$.

Therefore, if $\mathcal{A}$ wins, then $\mathcal{B}$ assumes it must be the real world, and distinguishes the two with the desired probability. $\square$

This result is unsurprising and not coincidental. Both definitions target the same security notion. However, when attempting to extend our constructions into the UC model described by Jarecki et al., we encounter an issue: without the final random oracle, we are unable to force the PRF-Srv queries to match up with the correct outputs.

On the other hand, we claim that wrapping the PRF-Srv response in a hash function as in the Jarecki et al. constructions, then the following is true:

Conjecture 1. Let Π be a partially oblivious PRF, H be a hash function modelled as a random oracle, and Π' be the functionality derived by matching the functionality of Π to a vOPRF. The final output is $H(t, m, F_k(t, m))$.

Then for all adversaries $\mathcal{A}$, we can construct an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\Pi'}^{\text{voprf}}(\mathcal{A}) \leq \left|\mathsf{Adv}_{\Pi}^{\text{om-unp}}(\mathcal{B}) - \frac{1}{2}\right|.$$

The intuition is that due to the unpredictability of Π, the random oracle outputs of H can be programmed to correspond to PRF outputs in the correct way.

3.7 Password Onions

Web servers and other systems frequently store passwords in hashed form. A *password onion* is the result of additionally invoking a PRF service to harden the hash. In currently suggested onions, one sequentially combines local hashing and application of the PRF service.

We now present a service that we have implemented on top of Pythia for managing password onions. First, we describe the limitations of contemporary systems as exemplified by a recently disclosed architecture employed by Facebook [150]. Then we show how our password-onion system, which was easily engineered on top of Pythia, can address these limitations.

In what follows, we use the term "client" or "web server" to denote the server performing authentication and storing derived values from passwords and "PRF server" to denote the Pythia service.

```
PW-Onion($pw$)
$h_1 \leftarrow \text{MD5}(pw)$
$sa \leftarrow_{\$} \{0,1\}^{160}$
$h_2 \leftarrow \text{HMAC[SHA-1]}(h_1, sa)$
$h_3 \leftarrow \text{PRF-Cl}(h_2) = \text{HMAC[SHA-256]}(h_2, msk)$
$h_4 \leftarrow \text{scrypt}(h_3, sa)$
$h_5 \leftarrow \text{HMAC[SHA-256]}(h_4)$
```

Figure 3.10: The Facebook password onion. PRF-Cl(h_2) invokes the Facebook PRF service HMAC[SHA-256](h_2, K_s) with PRF-service secret key K_s.

3.7.1 Facebook Password Onion

An example of a contemporary system, used by Facebook, is given in Figure 3.10. This figure is of "archaeological" interest. It appears that vulnerabilities in MD5 led to the addition of a layer of processing under SHA-1; when vulnerabilities were found in SHA-1, Facebook then added layers of SHA-256. As we explain later, full-blown replacement of MD5 and SHA-1 with SHA-256 was not easily accomplished. Their PRF service applies HMAC using a service-held secret and returns the result. In this architecture, an adversary that compromises the web server and the password hashes it stores must still mount an online attack against the PRF service to compromise accounts. This is a big advance on the hashing-only practices that are commonly used.

The Facebook architecture nevertheless has some shortcomings. It is easy to see from Figure 3.10 that Facebook's system, like most contemporary PRF services, lacks several important features present in Pythia. One is message privacy: the Facebook PRF service applies HMAC to h_2. This is the salted hash of the password, and so learning the salt as well as compromising the PRF service suffices to re-enable offline brute-force attacks. This threat is avoided by Pythia due to blinding.

Another feature is batch key updates. In fact, the Facebook PRF service does not permit autonomous key updates at all, in the sense of an update to msk that can be propagated into PRF output updates. Should the client (password database) be compromised, the only way to reconstitute a hash in an existing password onion is to *wait until a user logs in and furnishes pw*. It is not clear whether the Facebook PRF service performs granular rate-limiting, although no such capability is indicated in [149]. PYTHIA, as we shall see, addresses all of these issues by design in our password onion system.

The Facebook onion also presents a subtle performance issue. By applying cryptographic primitives serially, the time to hash a password equals the time for local computations, call it t_{local}, *plus* the time for the round-trip PRF service call, call it t_{prf}. An attacker that compromises the web service and PRF service incurs no network latency, and thus may gain a considerable advantage in guessing time over an honest web server. In our PYTHIA-based password onion service, we address this issue by observing that it is possible to avoid serialisation of key derivation functions on the web server and the PRF service call. That is, we introduce in our PYTHIA-based service the idea of *parallelisable password onions*.

3.7.2 Pythia Password Onion

The onion algorithm we construct for PYTHIA is shown in Figure 3.11. For PYTHIA, the output of PRF-Cl is an element of a group $\mathbb{G}_T$. To use this service, a web server stores (h, sa) upon password registration; it verifies a proffered password pw' by checking that $\text{UpParOnion}(w, sa, pw') = h$. Written out we have that:

$$h = u^z = e(H_1(sa), H_2(pw))^{k_w z}.$$

This design ensures that the key update functions in the PYTHIA API may be used to update onions as well. For example, to update an ensemble key k_w to k'_w, the service computes and furnishes to the web server an update token $\Delta_w = k'_w/k_w$. The web server may compute h^{Δ_w} for each stored value h.

The client also computes a password-based key-derivation function (PBKDF) of their own, representing the algorithm they would have traditionally used on the client-side. This ensures that even a full compromise of the PRF service and password database results in security degrading gracefully to that of the password hashing algorithm.

Parallelisation. Password verification here is *parallelisable* in the sense that z and u may be computed independently and then combined. Such parallel implementation

```
UpParOnion($w, sa, pw$)
$z \leftarrow \text{PBKDF}(pw, sa)$
$u \leftarrow \text{PRF-Cl}(w, sa, pw)$
$h \leftarrow u^z$
return $(h, sa)$
```

Figure 3.11: An updatable, parallelisable password onion. PRF-Cl returns elements of a group $\mathbb{G}$. The value w is a unique PRF-service identifier for the web server (e.g., a random 256-bit string) and sa is a random per-user salt value.

of the onion achieves a password verification latency of $\max\{t_{local}, t_{prf}\}$ (plus a single exponentiation), as opposed to $t_{local} + t_{prf}$ in a serialised implementation.

A web server generally aims to achieve a verification latency equal to some latency target T that is high enough to slow offline brute-force attacks, but low enough not to burden users. For a parallelised onion a web server can meet its latency target by setting $t_{local}, t_{prf} \approx T$. At the same time an offline attacker that has compromised the web server and Pythia must perform about $t_{local}+t_F > T$ work to check a single password guess, where t_F is the computation time of F_{k_w} (i.e., t_{prf} minus network latency). An attacker can parallelise, but her total work still goes up relative to the serial onion approach for the same latency target T.

We estimate the security improvement of parallel onions over serial onions using our benchmarks from Section 3.4.2. We fix a login latency budget of $T = 300\,\text{ms}$. This is the default setting for Python's bcrypt and scrypt modules, though all PBDKFs are tunable so one can choose T to be any value desired. The latency costs for a Pythia query with hot connections are 12 ms (LAN) and 96 ms (WAN). If one performs computations serially with a fixed T then PBKDF computations need to be reduced by 4% (LAN) and 32% (WAN) compared to the parallel approach. In the event that the Pythia server and password database are compromised, the serial onion enables speedup of offline dictionary attacks by the same percentages.

Rate limiting and logging. The transparency of tweaks enables the Pythia PRF service in this setting to execute any of a wide range of rate-limiting policies with per-account visibility (in contrast to what may be in Facebook an account-blind PRF service). As an example demonstrating the flexibility of our architecture, in our implementation Pythia performs a tiered rate-limiting: for a given account

(t), it limits queries to at most 10 per hour per account, and at most 300 per month. (In expectation, guessing a random 4-digit PIN would require 1.4 years under this policy.) It logs violations of these thresholds. In a production environment, it could also send alerts to security administrators.

We emphasize that a wide range of other rate-limiting policies is possible. We also point out that PYTHIA's rate limiting supplements that normally implemented at the web server for remote login requests. PYTHIA performs rate limiting and may issue alerts even if the web server is compromised.

Key update. The key update calls in the PYTHIA API, and the ability to rotate either k_w or msk efficiently, propagates up to the password onion service. Key updates instantly invalidate the web server's existing password database—a useful capability in case of compromise. A compromised database becomes useless to an attacker attempting to recover passwords, even with the ability to query PYTHIA. Using a key update token, the web server can then recover from compromise by refreshing its database.

We created a client simulator with MongoDB and the mongoengine Python module. With this we benchmarked key updates with 100,000 database entries. The client requested a key update from PYTHIA, received the update token Δ_w, and updated each database entry. The complete update required less than 1 ms per entry, and terminated in less than 97 seconds for all 100,000 entries. For a larger database we assume updates scale linearly, and so an update for 1 million users completes in under 17 minutes.

The web server need not lock the database to perform updates; it can execute them in parallel with normal login operations. Doing so does require additional versioning information for each entry to indicate the version of k_w (in the simplest form, whether or not it has received the latest update).

Database replication. Password databases can be replicated with a key transfer using the API call Transfer (see Section 3.5). In this replication each new copy uses a unique ensemble key selector and thus a cryptographically independent PRF service key. Given a database $w, \{(sa_1, h_1), \ldots, (sa_d, h_d)\}$ with d users, the administrator invokes $\mathsf{Transfer}(w, w')$ to obtain a token $\Delta_{w \to w'}$. The client computes $h'_i = h_i^{\Delta_{w \to w'}}$ for $i \in [1 \dots d]$ and sends the new database $(w', \{(sa_1, h'_1), \dots, (sa_d, h'_d)\})$ to the new server. The client does not modify salt values, which allows PYTHIA to link online guessing attacks carried out from multiple compromised web servers. Replication in this way costs database copy time plus 1 ms per entry to apply the update token, thus making it in the order of minutes for hundreds of thousands of users.

3.8 Hardened Brainwallets

Brainwallets are a common but dangerous way to secure accounts in the popular cryptocurrency Bitcoin, as well as in less popular cryptocurrencies such as Litecoin. Here we describe how the PYTHIA service can be used directly as a means to harden brainwallets. This application showcases the ease with which a wide variety of applications can be engineered around PYTHIA.

How brainwallets work. Every Bitcoin account has an associated private / public key pair (sk, pk). The private key sk is used to produce digital (ECDSA) signatures that authorise payments from the account. The public key pk permits verification of these signatures. It also acts as an account identifier; a Bitcoin address is derived by hashing pk (under SHA-256 and RIPEMD-160) and encoding it (in base 58, with a check value).

Knowledge of the private key sk equates with control of the account. If a user loses a private key, she therefore loses control over her account. For example, if a high entropy key sk is stored exclusively on a device such as a mobile phone or laptop, and the device is seized or physically destroyed, the account assets become irrecoverable.

Brainwallets offer an attractive remedy for such physical risks of key loss. A brainwallet is simply a password or passphrase P memorised by a Bitcoin account holder. The private key sk is generated directly from P. Thus the user's memory serves as the only instrument needed to authorise access to the account.

In more detail, the passphrase is typically hashed using SHA-256 to obtain a 256-bit string $sk = \text{SHA-256}(P)$. Bitcoin employs ECDSA signatures on the `secp256k1` elliptic curve; with high probability ($\approx 1 - 2^{-126}$), sk is less than the group order, and a valid ECDSA private key. (Some websites employ stronger key derivation functions. For example, WrapWallet by keybase.io [142] derives sk from an XOR of each of PBKDF2 and scrypt applied to P and permits use of a user-supplied salt.)

Since a brainwallet employs only P as a secret, and does not necessarily use any additional security measures, an attacker that guesses P can seize control of a user's account. As account addresses are posted publicly in the Bitcoin system (in the "blockchain"), an attacker can easily confirm a correct guess. Brainwallets are thus vulnerable to brute-force, offline guessing attacks. Numerous incidents have come to light showing that brainwallet cracking is a significant threat [64].

3.8.1 A Pythia-hardened Brainwallet

Pythia offers a simple, powerful means of protecting brainwallets against offline attack. Hardening P in the same manner as an ordinary password yields a strong key $\tilde{P}$ that can serve in lieu of P to derive sk.

To use Pythia, a user chooses a unique identifier id, e.g., her e-mail address, an account identifier $acct$, and a passphrase P. The identifier $acct$ might be used to distinguish among Bitcoin accounts for users who wish to use the same password for multiple wallets. The client then sends $(w = id, t = id \| acct, m = P)$ to the Pythia service to obtain the hardened value $F_{k_w}(t, m) = \tilde{P}$. Here, id is used both as an account identifier and as part of the salt. Message privacy in Pythia ensures that the service learns nothing about P. Then $\tilde{P}$ is hashed with SHA-256 to yield sk. The corresponding public key pk and address are generated in the standard way from sk [46].

Pythia forces a would-be brainwallet attacker to mount an online attack to compromise an account. Not only is an online attack much slower, but it may be rate-limited by Pythia and detected and flagged. As the Pythia service derives $\tilde{P}$ using a user-specific key, it additionally prevents an attacker from mounting a dictionary attack against multiple accounts. While in the conventional brainwallet setting, two users who make use of the same secret P will end up controlling the same account, Pythia ensures that the same password P produces distinct per-user key pairs.

Should an attacker compromise the Pythia service and steal msk and $\mathtt{K}$, the attacker must still perform an offline brute-force attack against the user's brainwallet. So in the worst case, a user obtains security with Pythia at least as good as without it.

Additional security issues. A few subtle security issues deserve brief discussion:

- *Stronger KDFs:* To protect against brute-force attack in the event of Pythia compromise, a resource-intensive key-derivation function may be desirable, as is normally used in password databases. This can be achieved by replacing the SHA-256 hash of $\tilde{P}$ above with an appropriate KDF computation, or alternatively using an onion approach described in Section 3.7.

- *Denial-of-service:* By performing rate-limiting, Pythia creates the risk of targeted denial-of-service attacks against Bitcoin users. As Bitcoin is pseudonymous, use of an e-mail address as a Pythia key-selector suffices to prevent such attacks against users based on their Bitcoin addresses alone. Users also have the option, of course, of using a semi-secret id. A general DoS attack against the Pythia service is also possible, but of similar concern for Bitcoin itself [47].

- *Key rotation:* Rotation of an ensemble key k_w (or the master key msk) induces a new value of $\tilde{P}$ and thus a new (sk, pk) pair and account. A client can handle such rotations in the naïve way: transfer funds from the old address to the new one.

- *Catastrophic failure of* Pythia*:* If a Pythia service fails catastrophically, e.g., msk or $\mathtt{K}$ is lost, then in a typical setting, it is possible simply to reset users' passwords. In the brainwallet case, the result would be loss of virtual-currency assets protected by the server—a familiar event for Bitcoin users [143]. This problem can be avoided, for instance, using a threshold implementation of Pythia, as mentioned in Section 3.8.2 or storing sk in a secure, offline manner like a safe-deposit box for disaster recovery.

3.8.2 Threshold Security

In order to gain both redundancy and security, we give a threshold scheme that can be used with a number of Pythia servers to protect a secret under a single password. This scheme uses Shamir's secret sharing threshold scheme [178] and gives (k, n) threshold security. That is, initially, n Pythia servers are contacted and used to protect a secret s, and then any k servers can be used to recover s and any adversary that has compromised fewer than k Pythia servers learns no information about s.

Preparation. The client chooses an ensemble key selector w, tweak t, password P, and contacts n Pythia servers to compute $q_i = \text{PRF-Cl}_i(w, t, P) \mod p$ for $0 < i \leq n$. The client selects a random polynomial of degree $k-1$ with coefficients from $\mathbb{Z}_p^*$ where p is a suitably large prime: $f(x) = \sum_{j=0}^{k-1} x^j a_j$. Let the secret $s = a_0$. Next the client computes the vector $\Phi = (\phi_1, ..., \phi_n)$ where $\phi_i = f(i) - q_i$. The client durably stores the value Φ, but does not need to protect it (it's not secret). The client also stores public keys p_i from each Pythia server to validate proofs when issuing future queries.

Recovery. The client can reconstruct s if she has Φ by querying any k Pythia servers giving k values q_i. These q_i values can be applied to the corresponding Φ values to retrieve k distinct points that lie on the curve $f(x)$. With k points on a degree $k-1$ curve, the client can use interpolation to recover the unique polynomial $f(x)$, which includes the curve's intercept $a_0 = s$.

Security. If an adversary is given Φ, w, t, the public keys p_i, a ciphertext based on s, and the secrets from $m < k$ Pythia servers, the adversary has no information that will permit her to verify password guesses offline. Compared to [178], this scheme reduces the problem of storing n secrets to having access to n secure OPRFs and durable (but non-secret) storage of the values Φ and public keys p_i.

Verification. Verification of server responses occurs within the Pythia protocol. If a server is detected to be dishonest (or goes out of service), it can be easily replaced by the client without changing the secret s. To replace a Pythia server that is suspected to be compromised or detected as dishonest, the client reconstructs the secret s using any k servers, executes Reset operations on all remaining servers: this effects a cryptographic erasure on the values Φ and $f(x)$. The client then selects a new, random polynomial, keeping a_0 fixed, and generates and stores an updated Φ' that maps to the new polynomial.

3.9 Related Work

We investigated a number of designs based on existing cryptographic primitives in the course of our work, though as mentioned none satisfied all of our design goals. Conventional PRFs built from block ciphers or hash functions fail to offer message privacy or key rotation. Consider instead the construction $H(t\|m)^{k_w}$ for $H : \{0,1\}^* \to \mathbb{G}$ a cryptographic hash function mapping onto a group $\mathbb{G}$. This was shown secure as a conventional PRF by Naor, Pinkas, and Reingold assuming decisional Diffie-Hellman (DDH) is hard in $\mathbb{G}$ and when modeling H as a random oracle [152]. It supports key rotations (in fact it is key-homomorphic [60]) and verifiability can be handled using non-interactive zero-knowledge proofs (ZKP) as in PYTHIA. But this approach fails to provide message privacy if we submit both t and m to the server and have it compute the full hash.

One can achieve message-hiding by using blinding: have the client submit $X = H(t\|m)^r$ for random $r \in \mathbb{Z}_{|\mathbb{G}|}$ and the server reply with X^{k_w} as well as a ZKP proving this was done correctly. The resulting scheme is originally due to Chaum and Pedersen [78], and suggested for use by Ford and Kaliski [107] in the context of threshold password-authenticated secret sharing (see also [19, 69, 137, 88]). There an end user interacts with one or more blind signature servers to derive a secret authentication token. If $\mathbb{G}$ comes equipped with a bilinear pairing, one can dispense with ZKPs. The resulting scheme is Boldyreva's blinded version [56] of BLS signatures [62]. However, neither approach provides granular rate limiting when blinding is used: the tweak t is hidden from the server. Even if the client sends t as well, the server cannot verify that it matches the one used to compute X and attackers can thereby bypass rate limits.

To fix this, one might use Ford-Kaliski with a separate secret key for each tweak. This would result in having a different key for each unique w, t pair. Message privacy is maintained by the blinding, and querying $w, t, H(t' \| m)^r$ for $t \neq t'$ does not help an attacker circumvent per-tweak rate limiting. But now the server-side storage grows in the number of unique w, t pairs, a client using a single ensemble w must now track N public keys when they use the service for N different tweaks, and key rotation requires N interactions with the PRF server to get N separate update tokens (one per unique tweak for which a PRF output is stored). When N is large and the number of ensembles w is small as in our password storage application, these inefficiencies add significant overheads.

Another issue with the above suggestions is that their security was only previously analysed in the context of one-more unforgeability [160] as targeted by blind signatures [77] and partially blind signatures [2]. (Some were analysed as conventional PRFs, but that is in a model where adversaries do not get access to a blinded server oracle.) The password onion application requires more than unforgeability because message privacy is needed. (A signature could be unforgeable but include the entire message in its signature, and this would obviate the benefits of a PRF service for most applications.) These schemes, however, can be proven to be one-more PRFs, the notion we introduce, under suitable one-more DDH style assumptions using the same proof techniques found in Section 3.6.

Fully oblivious PRFs [108] and their verifiable versions [117] also do not allow granular rate limiting. We note that the Jarecki, Kiayias, and Krawczyk constructions of verifiable OPRFs [117] in the RO model are essentially the Ford-Kaliski protocol above, but with an extra hash computation, making the PRF output $H'(t\|m\|H(t\|m)^{k_w})$. Our notion of one-more unpredictability captures the necessary requirements on the inner cryptographic component, and might modularise and simplify their proofs. Their transform is similar to the unique blind signature to OPRF transformation of Camenisch, Neven, and shelat [70]. None of these efficient oblivious PRF protocols support key rotations (with compact tokens or otherwise) as the final hashing step destroys updatability.

The setting of capture-resilient devices shares with ours the use of an off-system key-holding server and the desire to perform cryptographic erasure [136, 135]. They only perform protocols for encryption and signing functionalities, however, and not (more broadly useful) PRFs. They also do not support granular rate limiting and master secret key rotation.

Our security analysis only covers the core cryptographic components of Pythia, and future work would be needed to show the security of the API. Recent work relevant in this area is [179].

Our main construction coincides with prior ones for other contexts. The Sakai, Ohgishi, and Kasahara [174] identity-based non-interactive key exchange protocol computes a symmetric encryption key as $e(H_1(ID_1), H_2(ID_2))^k$ for k a master secret held by a trusted party and ID_1 and ID_2 being the identities of the parties. See [156] for a formal analysis of their scheme. Boneh and Waters suggest the same construction as a left-or-right constrained PRF [63]. The settings and their goals are different from ours, and in particular one cannot use either as-is for our applications. Naïvely one might hope that returning the constrained PRF key $H_1(t)^{k_w}$ to the client suffices for our applications, but in fact this totally breaks rate-limiting. Security analysis of our protocol requires new techniques, and in particular security must be shown to hold when the adversary has access to a half-blinded oracle — this rules out the techniques used in [156, 63].

Key-updatable encryption [60] and proxy re-encryption [54] both support key rotation, and could be used to encrypt password hashes in a way supporting compact update tokens and that prevents offline brute-force attacks. But this would require encryption and decryption to be handled by the hardening service, preventing message privacy.

Verifiable PRFs as defined by [145, 133, 92, 91] allow one to verify that a known PRF output is correct relative to a public key. Previous verifiable PRF constructions are not oblivious, let alone partially oblivious.

Threshold and distributed PRFs [146, 152, 91] as well as distributed key distribution centres [152] enable a sufficiently large subset of servers to compute a PRF output, but previous constructions do not provide the granular rate limiting and key rotation we desire. However, it is clear that there are situations where applications would benefit from a threshold implementation of Pythia, for both redundancy and distribution of trust, as discussed in Section 3.8.2 for the case of brainwallets.

Since the publication of our work, there have been a number of works expanding on the idea of using a partially-oblivious PRF for password hardening. Two particularly interesting works are by Schneider et al. [177], and by Lai et al. [125]. The former proposed weakening the requirement for the service to copy the functionality of a PRF, and instead considers a new primitive called partially-oblivious commitment schemes.

The work by Lai et al. in [125] extends both our work, and the previously described work by Schneider et al., resulting in a more comprehensive set of security notions for password hardening. Furthermore, [125] introduces a new password hardening construction capable of meeting these security notions, and furthermore satisfies the design goals set forwards in our original work.

However, one limitation of both of these approaches, is the lack of flexibility. Although password hardening is one particularly attractive use case for a PRF service such as Pythia, the existence of a generic PRF service could potentially have greater use cases in the future than initially envisioned.

3.10 Conclusion

We presented the design and implementation of Pythia, a modern PRF service. Prior works have explored the use of remote cryptographic services to harden keys derived from passwords or otherwise improve resilience to compromise. Pythia, however, transcends existing designs to simultaneously support granular rate limiting, efficient key rotation, and cryptographic erasure. This set of features, which stems from practical requirements in applications such as enterprise password storage, proves to require a new cryptographic primitive that we refer to as a partially oblivious PRF.

Unlike a (fully) oblivious PRF, a partially oblivious PRF causes one portion of an input to be revealed to the server to enable rate limiting and detection of online brute-force attacks. We provided a bilinear-pairing based construction for partially oblivious PRFs that is highly efficient and simple to implement (given a pairings library), and also supports efficient key rotations. A formal proof of security is unobtainable using existing techniques (such as those developed for fully oblivious PRFs). We thus gave new definitions and proof techniques that may be of independent interest.

3.10 Conclusion

We implemented Pythia and show how it may be easily integrated it into a range of applications. We designed a new enterprise “password onion” system that improves upon the one recently reported in use at Facebook. Our system permits fast key rotations, enabling practical reactive and proactive key management, and uses a parallelisable onion design which, for a given authentication latency, imposes more computational effort on attackers after a compromise. We also explored the use of Pythia to harden brainwallets for cryptocurrencies.

Chapter 4

Key Rotation for Authenticated Encryption

Contents

In this chapter, we study the area of key rotation for encrypted data, stored by a third party. We take a twofold approach, developing new security notions strengthening prior work, and propose a new updatable encryption scheme which achieves our strongest notions.

The proposed security notions are the result of a careful balancing process, on one side is the desire to produce the strongest possible security notion, while on the other side we balance against realistic concerns, and achievable goals.

4.1 Introduction

To cryptographically protect data while stored, systems use authenticated encryption (AE) schemes that provide strong message confidentiality as well as ciphertext integrity. The latter allows detection of active attackers who manipulate ciphertexts. When data is stored for long periods of time, good key management practice dictates that systems must support key rotation: moving encrypted data from an old key to a fresh one. Indeed, key rotation is mandated by regulation in some contexts, such as the payment card industry data security standard (PCI DSS) that dictates how credit card data must be secured [157]. Key rotation can also be used to revoke old keys that are compromised, or to effect data access revocation.

Deployed approaches to key rotation. Systems used in practice typically support a type of key rotation using a symmetric key hierarchy. Amazon's Key Management Service [18], for example, enables users to encrypt a plaintext M under a fresh data encapsulation key via $C_{dem} = \mathsf{Enc}(K_d, M)$ and then wrap K_d via $C_{kem} = \mathsf{Enc}(K, K_d)$ under a long-term key K owned by the client. Here Enc is an authenticated encryption (AE) scheme. By analogy with the use of hybrid encryption in the asymmetric setting, we refer to such a scheme as a KEM/DEM construction, with KEM and DEM standing for key and data encapsulation mechanisms, respectively; we refer to the specific scheme as AE-hybrid.

The AE-hybrid scheme then allows a simple form of key rotation: the client picks a fresh K' and re-encrypts K_d as $C'_{kem} = \mathsf{Enc}(K', \mathsf{Dec}(K, C_{kem}))$. Note that the DEM key K_d does not change during key rotation. When deployed in a remote storage system, a client can perform key rotation just by fetching from the server the small, constant-sized ciphertext C_{kem}, operating locally on it to produce C'_{kem}, and then sending C'_{kem} back to the server. Performance is independent of the actual message length. The Google Cloud Platform [112] uses a similar approach to enable key rotation.

To our knowledge, the level of security provided by this widely deployed AE-hybrid scheme has never been investigated, let alone formally defined in a security model motivated by real-world security considerations. It is even arguable whether AE-hybrid truly rotates keys, since the DEM key does not change. Certainly it is unclear what security is provided if key compromises occur, one of the main motivations for using such an approach in the first place. On the other hand, the scheme is fast and requires only limited data transfer between the client and the data store, and appears to be sufficient to meet current regulatory requirements.

Updatable encryption. Boneh, Lewi, Montgomery, and Raghunathan (BLMR) [61] (the full version of [60]) introduced another approach to enabling key rotation that they call *updatable encryption*. An updatable encryption scheme is a symmetric encryption scheme that, in addition to the usual triple of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ algorithms, comes with a pair of algorithms $\mathsf{ReKeyGen}$ and ReEnc. The first, $\mathsf{ReKeyGen}$, generates a compact rekey token given the old and new secret keys and a target ciphertext, while the second, ReEnc, uses a rekey token output by the first to rotate the ciphertext without performing decryption. For example, AE-hybrid can be seen as an instance of an updatable encryption scheme in which the rekey token output by $\mathsf{ReKeyGen}$ is C'_{kem} and where ReEnc simply replaces C_{kem} with C'_{kem}. BLMR introduced an IND-CPA-style security notion in which adversaries can additionally obtain some rekey tokens. Their definition is inspired by, but different from, those used for CCA-secure proxy re-encryption schemes [73]. Given its obvious limitations when it comes to key rotation, it is perhaps surprising that the AE-hybrid construction provably meets the BLMR confidentiality notion for updatable encryption schemes.

BLMR also introduced and targeted a second security notion for updatable encryption, called ciphertext independence. It demands that a ciphertext and its rotation to another key are identically distributed to a ciphertext and a rotation of another ciphertext (for the same message). The intuition is that this captures the idea that true key rotation should refresh all randomness used during encryption. This definition is *not* met by the AE-hybrid construction above. But it is both unclear what attacks meeting their definition would prevent, and, relatedly, whether more intuitive definitions exist.

BLMR gave a construction for an updatable encryption scheme and claimed that it provably meets their two security definitions. Their construction cleverly combines an IND-CPA KEM with a DEM that uses a key-homomorphic PRF [152, 61] to realise a stream cipher. This enables rotation of both the KEM and the DEM keys, though the latter requires a number of operations that is linear in the plaintext length. Looking ahead, their proof sketch has a bug and we provide strong evidence that it is unlikely to be fixable. Moreover, BLMR do not yet target or achieve any kind of authenticated encryption goal, a must for practical use.

Our contributions. We provide a systematic treatment of AE schemes that support key rotation without decryption, a.k.a. updatable AE.

Specifically, we provide a new security notion for confidentiality, UP-IND, that is strictly stronger than that of BLMR [61], a corresponding notion for integrity, UP-INT (missing entirely from BLMR but essential for practice), and a new notion called re-encryption indistinguishability (UP-REENC) that is strictly stronger and more natural in capturing the spirit of "true key rotation" than the ciphertext indistinguishability notion of BLMR.

Achieving our UP-REENC notion means that an attacker, having access to both a ciphertext and the secret key used to generate it, should not be able to derive any information that helps it attack a rotation of that ciphertext. Thus, for example, an insider with access to the encryption keys at some point in time but who is then excluded from the system cannot make use of the old keys to learn anything useful once key rotation has been carried out on the AE ciphertexts. Teasing out the correct form of this notion turns out to be a significant challenge in our work.

Armed with this set of security notions, we go on to make better sense of the landscape of constructions for updatable AE schemes. Table 4.1 summarises the security properties of the different schemes that we consider. Referring to this table, our security notions highlight the limitations of the AE-hybrid scheme: while it meets the confidentiality notion of BLMR, it only satisfies our UP-IND and UP-INT notions when considering a severely weakened adversary who has no access to any compromised keys. We propose an improved construction, KSS, that satisfies both notions for any number of compromised keys and which is easily deployable via small adjustments to AE-hybrid. KSS uses a form of secret sharing to embed key shares in the KEM and DEM components to avoid the issue of leaking the DEM key in the updating process, and adds a cryptographic hash binding the KEM and DEM components to prevent mauling attacks. These changes could easily be adopted by practitioners with virtually no impact on performance, while concretely improving security.

However, the improved scheme KSS cannot satisfy our UP-REENC notion, because it still uses a KEM/DEM-style approach in which the DEM key is never rotated. The BLMR scheme might provide UP-REENC security, but, as noted above, its security proof contains a bug which we consider unlikely to be fixable. Indeed, we show that proving the BLMR scheme confidential would imply that one could also prove circular security [48, 68] for a particular type of hybrid encryption scheme assuming only the key encapsulation is IND-CPA secure. Existing counter-examples of IND-CPA secure, but circular insecure, schemes [3, 76] do not quite rule out such a result. But the link to the very strong notion of circular security casts doubt on the security of this scheme. One can easily modify the BLMR scheme to avoid this issue, but even having done so the resulting encryption scheme is still trivially malleable and so cannot meet our UP-INT integrity notion.

We therefore provide another new scheme, ReCrypt, meeting all three of our security notions: UP-IND, UP-INT and UP-REENC. We take inspiration from the previous constructions, especially that of BLMR: key-homomorphic PRFs provide the ability to fully rotate encryption keys; the KEM/DEM approach with secret sharing avoids the issue of leaking the DEM key in the updating process; and finally, adding a cryptographic hash to the KEM tightly binds the KEM and DEM portions and prevents ciphertext manipulation. We go on to instantiate the scheme using the Random Oracle Model (ROM) key-homomorphic PRF from [152], having the form $H(M)^k$, where H is a hash function into a group in which DDH is hard. This yields a construction of an updatable AE scheme meeting all three of our security notions in the ROM under the DDH assumption. We report on the performance of an implementation of ReCrypt using elliptic curve groups, concluding that it is performant enough for practical use with short plaintexts. However, because of its reliance on exponentiation, ReCrypt is still orders of magnitude slower than our KSS scheme (achieving only UP-IND and UP-INT security). This, currently, is the price that must be paid for true key rotation in updatable encryption.

Summary. In summary, the main contributions of this chapter are:

- To provide the first definitions of security for AE supporting key rotation without exposing plaintext.
- To explain the gap between existing, deployed schemes using the KEM/DEM approach and "full" refreshing of ciphertexts.
- To provide the first proofs of security for AE schemes using the KEM/DEM approach, namely AE-hybrid and KSS.

Scheme	Section	Tokens	CT dep.	UP-IND	UP-INT	UP-REENC
AE-hybrid†	4.5.1	uni-dir.	depend.	✗	✗	✗
KSS*	4.5.3	uni-dir.	depend.	✓	✓	✗
XOR-KEM*	4.6	bi-dir.	indep.	✓	✗	✗
BLMR	4.8	uni-dir.	depend.	✗	✗	✗
ReCrypt*	4.9	uni-dir.	depend.	✓	✓	✓

Table 4.1: Summary of schemes studied. † In-use by practitioners today. * Introduced in this work.

- To detail the first updatable AE scheme, ReCrypt, that fully and securely refreshes ciphertexts by way of key rotations without exposing plaintext. We implement a prototype and report on microbenchmarks, showing that rotations can be performed in less than 10 μs per byte.

4.2 Updatable AE

We turn to formalising the syntax and semantics of AE schemes supporting key rotation. Our approach extends that of Boneh et al. [61] (BLMR), the main syntactical difference being that we allow rekey token generation, re-encryption, and decryption to all return a distinguished error symbol $\perp$. This is required to enable us to later cater for integrity notions. We also modify the syntax so that ciphertexts include two portions, a header and a body. In our formulation, only the former is used during generation of rekey tokens (while in BLMR the full ciphertext is formally required).

Definition 13 (Updatable AE)**.** An *updatable AE scheme* is a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$ with the following properties:

- $\mathsf{KeyGen}() \to k$. Outputs a secret key k.
- $\mathsf{Enc}(k, m) \to C$. On input a secret key k and message m, outputs a ciphertext $C = (\tilde{C}, \overline{C})$ consisting of a ciphertext header $\tilde{C}$ and ciphertext body $\overline{C}$.
- $\mathsf{ReKeyGen}(k_1, k_2, \tilde{C}) \to \Delta_{1,2,\tilde{C}}$. On input two secret keys, k_1 and k_2, and a ciphertext header $\tilde{C}$, outputs a rekey token or $\perp$.
- $\mathsf{ReEnc}(\Delta_{1,2,\tilde{C}}, (\tilde{C}, \overline{C})) \to C_2$. On input a rekey token and ciphertext, outputs a new ciphertext or $\perp$. We require that ReEnc is deterministic.

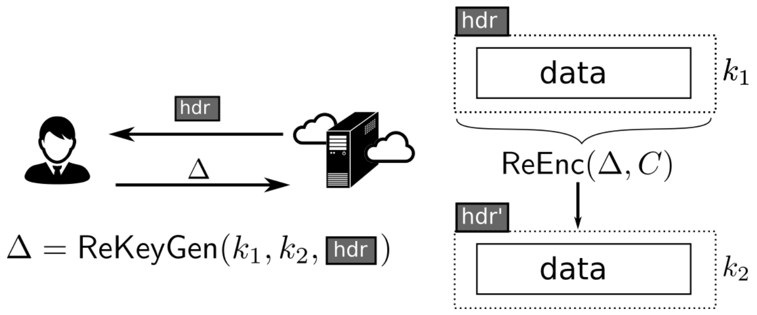

Figure 4.1: Interaction between client and cloud during a ciphertext-dependent update. Client retrieves a small ciphertext header, and runs ReKeyGen to produce a compact rekey token Δ. The cloud uses this token to re-encrypt the data. At the end of the update, the data is encrypted using k_2, and cannot be recovered using only k_1.

- $\mathsf{Dec}(k, C) \to m$. On input a secret key k and ciphertext C outputs either a message or $\perp$.

Of course we require that all algorithms are efficiently computable. Note that, in common with [61], our definition is *not* in the nonce-based setting that is widely used for AE. Rather, we will assume that Enc is randomised. We consider this sufficient for a first treatment of updatable AE; it also reflects common industry practice as per the schemes currently used by Amazon [18] and Google [112]. We relegate the important problem of developing a parallel formulation in the nonce-based setting to future work. Similarly, we assume that all our AE schemes have single decryption errors, cf. [57], and we do not consider issues such as release of unverified plaintext, cf. [11], tidiness, cf. [151] and length-hiding, cf. [155].

Correctness. An updatable AE scheme is *correct* if decrypting a legitimately generated ciphertext reproduces the original message. Of course, legitimate ciphertexts may be rotated through many keys, complicating the formalisation of this notion.

Definition 14 (Correctness)**.** Fix an updatable AE scheme Π. For any message m and any sequence of secret keys $k_1, \ldots k_T$ output by running KeyGen T times, let $C_1 = (\tilde{C}_1, \overline{C}_1) = \mathsf{Enc}(k_1, m)$ and recursively define for $1 \le t < T$

$$C_{t+1} = \mathsf{ReEnc}(\mathsf{ReKeyGen}(k_t, k_{t+1}, \tilde{C}_t), C_t).$$

Then Π is correct if $\mathsf{Dec}(k_T, C_T) = m$ with probability 1.

Compactness. We say that an updatable AE scheme is compact if the size of both ciphertext headers and rekeying tokens are independent of the length of the plaintext. In practice the sizes should be as small as possible, and for the constructions we consider these are typically a small constant multiple of the key length.

Compactness is important for efficiency of key rotation. Considering the abstract architecture in Figure 4.1, header values must be available to the key server when rekey tokens are generated. Typically this will mean having to fetch them from storage. Likewise, the rekey token must be sent back to the storage system. Note that there are simple constructions that are not compact, such as the one that sets $\tilde{C}$ to be a standard authenticated encryption of the message and in which $\mathsf{ReKeyGen}$ decrypts $\tilde{C}$, re-encrypts it, and outputs a "rekeying token" as the new ciphertext.

Ciphertext-dependence. As formulated above, updatable AE schemes require part of the ciphertext, the ciphertext header $\tilde{C}$, in order to generate a rekey token. We will also consider schemes for which $\tilde{C}$ is the empty string, denoted ε. We will restrict attention to schemes for which encryption either always outputs $\tilde{C} = \varepsilon$ or never does. In the former case we call the scheme ciphertext-independent and, in the latter case, ciphertext-dependent. When discussing ciphertext-independent schemes, we will drop $\tilde{C}$ from notation, e.g., writing $\Delta_{i,j}$ instead of $\Delta_{i,j,\tilde{C}}$.

However, we primarily focus on ciphertext-dependent schemes which appear to offer more flexibility and achieve stronger security guarantees (though it is an open question whether a ciphertext-independent scheme can achieve our strongest security notion). We do propose a very lightweight ciphertext-independent scheme included in Section 4.6, but we show it achieves strictly weaker confidentiality and integrity notions. One can generically convert a ciphertext-independent scheme into a ciphertext-dependent one, simply by deriving a ciphertext-specific key using some unique identifier for the ciphertext. We omit the formal treatment of this trivial approach.

Directionality of rotations. Some updatable AE schemes are bidirectional, meaning rekey tokens can be used to go forwards or backwards.

We only consider bi-directionality to be a feature of ciphertext-independent schemes. Formally, we say that a scheme is *bidirectional* if there exists an efficient algorithm $\mathsf{Invert}(\cdot)$ that produces a valid rekey token $\Delta_{j,i}$ when given $\Delta_{i,j}$ as input.

Schemes that are not bidirectional might be able to ensure that an adversary cannot use rekey tokens to "undo" a rotation of a ciphertext. We will see that ciphertext-dependence can help in building such unidirectional schemes, whereas ciphertext-independent schemes seem harder to make unidirectional. This latter difficulty is related to the long-standing problem of constructing unidirectional proxy re-encryption schemes in the public key setting.

Relationship to proxy re-encryption. Proxy re-encryption targets a different setting than updatable encryption (or AE): the functional ability to allow a ciphertext encrypted under one key to be converted to a ciphertext decryptable by another key. The conversion should not leak plaintext data, but, unlike key rotation, it is not necessarily a goal of proxy re-encryption to remove all dependency on the original key, formalised as indistinguishability of re-encryptions (UP-REENC security) in our work. For example, previous work [80, 115] suggests twice encrypting plaintexts under different keys. To rotate, the previous outer key and a freshly generated outer key is sent to the proxy to perform conversion, but the inner key is never modified. Such an approach does not satisfy the goals of key rotation.

That said, any bidirectional, ciphertext-independent updatable AE ends up also being usable as a symmetric proxy re-encryption scheme (at least as formalised by [61]).

4.3 Motivating Scenarios

Fundamentally, updatable encryption schemes should maintain at minimum the same level of security as regular authenticated encryption schemes. However, when proposing security notions supporting key rotation, we need to consider the various parties in the protocol, and possible compromise scenarios.

Here we detail some motivating examples to provide an intuition of desirable security properties. We elaborate on a number of concerns which an effective key rotation must address. For each of these requirements, we identify where current mechanisms are lacking, highlighting the need for a new encryption primitive.

4.3.1 Untrusted Server

In general, cloud service providers are trusted to act honestly in accordance with their contractual obligations. A completely malicious service provider who simply deletes all encrypted data is not considered within the scope of updatable encryption.

We consider the storage service to be a protocol participant who follows the protocol honestly, but may attempt to infer additional information about, for example, the plaintext or secret keys used by the client.

Therefore, we wish to capture attacks in which information is revealed either directly, or indirectly, from the cloud storage provider. This includes, for example, the multitude of attacks ([166, 13, 183] to name a few) which are capable of retrieving data used in co-located virtual machines. It also includes cloud providers that have been compromised or compelled by law to reveal client information. As such, our resultant security model permits the attacker access to legitimate ciphertexts.

This setting rules out trivial implementations which rely on the cloud to perform encryption and decryption.

4.3.2 Client-Side Security

One consideration for performing proactive key rotations is whether the additional risk introduced by the key rotation mechanism outweighs the risk present by using the same cryptographic keys for a long time. This is a key principle in [22].

Therefore, it is important that the key rotation mechanism requires a minimal amount of cryptographic computation by the client. Given that the client has already shown an affinity for outsourcing resources to the cloud, it is unlikely that they will have the desire nor resources to perform this computation securely.

For example, using the naive re-encryption method – decrypting the entire ciphertext and re-encrypting locally – results in regular intervals at which the data will be residing as plaintext. If a large corpus of ciphertext must be updated (megabytes or more), performing all computations inside a hardware security module is likely not feasible.

When designing our security models, we include the ability for the adversary to play the role of the client in a key rotation, receiving all the information from the server that the client would typically receive.

4.3.3 Key Revocation

There are many reasons for why key rotation is an important part of the long-term security of encrypted data. Some examples can be found in [22]. In general, these can be divided into two distinct concerns: to reduce the impact of compromised or disclosure of keys; and to revoke legitimate access to data. Both scenarios require recovering security after a particular party has had access to cryptographic keys used.

A motivating example for this is used in [173]:

> Consider an employee with access to sensitive documents necessary for his work. One day, this employee is terminated and has his access revoked. Now, this employee with insider knowledge of the organization's systems, and who has retained his old key, may attempt to penetrate the database server and decrypt all the files that he once had access to.

Following on from the construction given in the introduction, and looking ahead to Section 4.5, consider an updatable encryption scheme which uses a authenticated encryption scheme $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and encrypts data via $\mathsf{Enc}(k_1, m) = (\mathcal{E}(k_1, x), \mathcal{E}(x, m))$. That is, a fresh data-encryption key x is generated on each encryption, which encrypts the message and is itself encrypted by the long-term key k_1.

As we will see in Section 4.5, this scheme permits key rotation by simply re-encrypting the encapsulated key x.

Therefore, an updated ciphertext is of the form $(\mathcal{E}(k_2, x), \mathcal{E}(x, m))$. It is clear that an adversary only with knowledge of k_2 cannot recover the message m.

However, a notable shortcoming of this scheme is that *temporary* access to a ciphertext while in possession of the key allows the adversary to exfiltrate the encapsulated key x. Since this value x is compact - usually linear in the security parameter - it is significantly easier to store the value x instead of the ciphertext.

Furthermore, knowledge of x is sufficient to decrypt and retrieve the message m at a later date, regardless of any subsequent re-encryptions. In practice, this scenario is a realistic concern, since the client is unable to recover security after a temporary breach.

This shows that while it is relatively easy to construct a scheme which appears to perform key revocation efficiently, a more nuanced definition is needed to capture the realities of key revocation scenarios.

This example forms the core of our strongest notion of updatable encyption, further detailed in Section 4.7, and this example forms the basis of many of our constructions.

Put together, these examples suggest a security model in which the adversary can compromise both clients and servers, with access to keys and ciphertexts. As we will see in the following section, capturing these abilities requires subtle definitions of security.

4.4 Confidentiality and Integrity for Updatable Encryption

Updatable AE should provide confidentiality for messages as well as integrity of ciphertexts, even in the face of adversaries that obtain rekey tokens and re-encryptions, and that can corrupt some number of secret keys. Finding definitions that rule out trivial wins — e.g., rotating a challenge ciphertext to a compromised key, or obtaining sequences of rekey tokens that allow such rotations — is delicate. We provide a framework for doing so.

Our starting point will be a confidentiality notion which improves significantly upon the previous notion of BLMR by including additional attack vectors, and strengthening existing ones.

For ciphertext integrity, we develop a new definition, building on the usual INT-CTXT notion for standard AE [31]. Looking ahead, we will target unidirectional schemes that simultaneously achieve both UP-IND and UP-INT security.

We will follow a concrete security approach in which we do not strictly define security, but rather measure advantage as a function of the resources (running time and number of queries) made by an adversary. Informally, schemes are secure if no adversary with reasonable resources can achieve advantage far from zero.

4.4.1 Message Confidentiality

The confidentiality game UP-IND is shown in the leftmost column of Figure 4.2. The adversary's goal is to guess the bit b. Success implies that a scheme leaks partial information about plaintexts. We parametrise the game by two values t and κ. The game initialises $t + \kappa$ secret keys, κ of which are given to the adversary, and t are kept secret for use in the oracles. We label the keys by $k_1, \ldots, k_t$ for the uncompromised keys, and by $k_{t+1}, \ldots k_{t+\kappa}$ for the compromised keys. We require at least one uncompromised key, but do not necessarily require any compromised keys, i.e. $t \geq 1$ and $\kappa \geq 0$. We leave consideration of equivalences between models with many keys and few keys and between models with active and static key compromises as interesting problems for future work.

The game relies on two subroutines $\mathsf{Invalid}_{\mathsf{RK}}$ and $\mathsf{Invalid}_{\mathsf{RE}}$ to determine if a rekeygen and re-encryption query, respectively, should be allowed. These procedures are efficiently computed by the game as a function of the adversarial queries and responses. This reliance on the transcript we leave implicit in the notation to avoid clutter. Different choices of invalidity procedures gives rise to distinct definitions of security, and we explain two interesting ones in turn. Note that an invalid query (as determined by $\mathsf{Invalid}_{\mathsf{RE}}$) still results in the adversary learning the ciphertext header, giving greater power to the adversary. We believe this to be an important improvement both in practice and theoretically over previous models, which consider only a partial compromise. The full compromise of a client results in the adversary playing the role of the client in the key update procedure, during which the server will return the ciphertext header. In practice, it is likely that an adversary who has initially breached the client would use this access to query related services.

```
UP-IND
$b \leftarrow_\$ \{0,1\}$
$k_1, \ldots, k_{t+\kappa} \leftarrow_\$ \mathsf{KeyGen}()$
$b' \leftarrow_\$ \mathcal{A}^O(k_{t+1}, \ldots, k_{t+\kappa})$
return $(b' = b)$

Enc$(i, m)$
return $\mathsf{Enc}(k_i, m)$

ReKeyGen$(i, j, \tilde{C})$
if $\mathsf{Invalid_{RK}}(i, j, \tilde{C})$ then return $\perp$
$\Delta_{i,j,\tilde{C}} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
return $\Delta_{i,j,\tilde{C}}$

ReEnc$(i, j, (\tilde{C}, \overline{C}))$
$\Delta_{i,j,\tilde{C}} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
$C' = (\tilde{C}', \overline{C}') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \overline{C}))$
if $\mathsf{Invalid_{RE}}(i, j, \tilde{C})$ then return $\tilde{C}'$
else return $C'$

LR$(i, m_0, m_1)$
if $i > t$ then return $\perp$
$C \leftarrow_\$ \mathsf{Enc}(k_i, m_b)$
return $C$
```

```
UP-INT
$\mathsf{win} \leftarrow \mathsf{false}$
$k_1, \ldots, k_{t+\kappa} \leftarrow_\$ \mathsf{KeyGen}()$
$\mathcal{A}^O(k_{t+1}, \ldots, k_{t+\kappa})$
return $\mathsf{win}$

Enc$(i, m)$
return $\mathsf{Enc}(k_i, m)$

ReKeyGen$(i, j, \tilde{C})$
return $\mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$

ReEnc$(i, j, (\tilde{C}, \overline{C}))$
$\Delta_{i,j,\tilde{C}} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
$C' \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \overline{C}))$
return $C'$

Try$(i, C)$
if $\mathsf{InvalidCTXT}(i, C)$ then return $\perp$
$M \leftarrow \mathsf{Dec}(k_i, C)$
if $M = \perp$ then return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
return $M$
```

Figure 4.2: Confidentiality and integrity games for updatable encryption security.

Invalidity procedures. For the invalidity constraints used in UP-IND, we target a strong definition, while preventing the adversary from trivially receiving a challenge ciphertext re-encrypted to a compromised key.

We use the ciphertext headers to determine whether a ciphertext has been derived from a challenge ciphertext. It is natural to use only the headers since these will be processed by the client when performing an update. We define a procedure $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C})$ that outputs true should $\tilde{C}$ have been derived from the ciphertext header returned by an LR query.

Definition 15 (LR-derived headers)**.** We recursively define function $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C})$ to output true iff any of the following conditions hold:

- $\tilde{C}$ was the ciphertext header output in response to a query $\mathrm{LR}(i, m_0, m_1)$
- $\tilde{C}$ was the ciphertext header output in response to a query $\mathrm{ReEnc}(j, i, C')$ and $\mathsf{Derived}_{\mathsf{LR}}(j, \tilde{C}') = \mathsf{true}$
- $\tilde{C}$ is the ciphertext header output by running $\mathsf{ReEnc}(\Delta_{j,i,\tilde{C}'}, C')$ where $\Delta_{j,i,\tilde{C}'}$ is the result of a query $\mathrm{ReKeyGen}(j, i, \tilde{C}')$ for which $\mathsf{Derived}_{\mathsf{LR}}(j, \tilde{C}') = \mathsf{true}$.

The predicate $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C})$ is efficient to compute and can be computed locally by the adversary. The most efficient way to implement it is to grow a look-up table $\mathtt{T}$, indexed by a key identifier and a ciphertext header, whose entries are sets of ciphertexts. Any query to $\mathrm{LR}(i, m_0, m_1)$ updates the table by adding the returned ciphertext to the set $\mathtt{T}[i, \tilde{C}]$ where $\tilde{C}$ is the oracle's returned ciphertext header value. For a query $\mathrm{ReEnc}(j, i, C')$, if $\mathtt{T}[j, \tilde{C}']$ is not empty, then it adds the returned ciphertext to the set $\mathtt{T}[i, \tilde{C}^*]$ for $\tilde{C}^*$ the returned ciphertext header. For a query $\mathrm{ReKeyGen}(j, i, \tilde{C}')$ with return value $\Delta_{j,i,\tilde{C}'}$, apply $\mathsf{ReEnc}(\Delta_{j,i,\tilde{C}'}, C)$ for all ciphertexts C found in entry $\mathtt{T}[j, \tilde{C}']$ and add appropriate new entries to the table. In this way, one can maintain the table in worst-case time that is quadratic in the number of queries and compute in constant time $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C})$ by simply checking if $\mathtt{T}[i, \tilde{C}]$ is non-empty. If any call to $\mathsf{ReKeyGen}$ or ReEnc in $\mathsf{Derived}_{\mathsf{LR}}$ or the main oracle procedure returns $\perp$, then the entire procedure returns $\perp$.

Note that $\mathsf{Derived}_{\mathsf{LR}}$ relies on ReEnc being deterministic, a restriction we made in Section 4.2. To complete the definition, we specify the invalidity procedures that use $\mathsf{Derived}_{\mathsf{LR}}$ as a subroutine:

- $\mathsf{Invalid}_{\mathsf{RK}}(i, j, \tilde{C})$ outputs true if $j > t$ and $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C}) = \mathsf{true}$. In words, the target key is compromised and $i, \tilde{C}$ derives from an LR query.
- $\mathsf{Invalid}_{\mathsf{RE}}(i, j, \tilde{C})$ outputs true if $j > t$ and $\mathsf{Derived}_{\mathsf{LR}}(i, \tilde{C}) = \mathsf{true}$. In words, the target key is compromised and $i, \tilde{C}$ derives from an LR query.

We denote the game defined by using these invalidity procedures by UP-IND. We associate to an UP-IND adversary $\mathcal{A}$ and scheme Π the advantage measure:

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-IND}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1 \,.$$

This notion is very strong and bidirectional schemes cannot meet it.

Theorem 5. *Let Π be a bidirectional updatable encryption scheme. Then there exists an* UP-IND *adversary $\mathcal{A}$ that makes two queries and for which*

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) = 1,$$

for any $\kappa \geq 1$ and $t \geq 1$.

Proof. We explicitly define the adversary $\mathcal{A}$. It makes a query to $C_1 = \text{LR}(1, m_0, m_1)$ for arbitrary messages $m_0 \neq m_1$ and computes locally $C_{t+1} = \mathsf{Enc}(k_{t+1}, m_1)$. It then makes a query $\Delta_{t+1,1,\tilde{C}_{t+1}} = \text{ReKeyGen}(t+1, 1, \tilde{C}_{t+1})$. It computes $C' = \mathsf{ReEnc}(\mathsf{Invert}(\Delta_{t+1,1,\tilde{C}_{t+1}}, C_{t+1}, C_1), C_1)$ locally and then decrypts C' using k_{t+1}. It checks whether the result is m_0 or m_1 and returns the appropriate bit. □

BLMR confidentiality. In comparison, we define invalidity procedures corresponding to those in BLMR's security notion.

- $\mathsf{InvalidBLMR}_{\mathsf{RK}}(i, j, \tilde{C})$ outputs true if $i \leq t < j$ or $j \leq t < i$ and outputs false otherwise. In words, the query is not allowed if exactly one of the two keys is compromised.

- $\mathsf{InvalidBLMR_{RE}}(i, j, \tilde{C})$ outputs true if $j > t$ and false otherwise. In words, the query is not allowed if the target key k_j is compromised.

We denote the game defined by using these invalidity procedures by UP-IND-BI (the naming will become clear presently). We associate to an UP-IND-BI adversary $\mathcal{A}$, scheme Π, and parameters κ, t the advantage measure:

$$\mathsf{Adv}^{\text{up-ind-bi}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-IND-BI}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1\,.$$

A few observations are in order. First, it is apparent that the invalidity procedures for the BLMR notion are significantly stronger than ours, leading to a weaker security notion: the BLMR procedures are not ciphertext-specific but instead depend only on the compromise status of keys. We will show that this difference is significant. In addition, the corresponding BLMR definition did not consider leakage of the ciphertext header when $\mathsf{InvalidBLMR_{RE}}$ returns true. Second, for ciphertext-independent schemes in which $\tilde{C} = \varepsilon$ always, the BLMR definition coincides with symmetric proxy re-encryption security (as also introduced in their paper [61]). Third, the BLMR confidentiality notion does not require unidirectional security of rekey tokens because it has the strong restriction of disallowing attackers from obtaining rekey tokens $\Delta_{i,j,\tilde{C}}$ with $i > t$ (so the corresponding key is compromised), but with $j < t$ (for an uncompromised key). Thus, in principle, bidirectional schemes could meet this notion, explaining our naming convention for the notion. Finally, the BLMR notion does not require ciphertext-specific rekey tokens because the invalidity conditions are based only on keys and not on the target ciphertext.

Detailed in Section 4.6 is a bidirectional scheme that is secure in the sense of UP-IND-BI. This result and the negative result that no bidirectional scheme can achieve UP-IND given above (Theorem 5) yields as a corollary that UP-IND-BI security is strictly weaker than UP-IND security. This illustrates the enhanced strength of our UP-IND security notion compared to the corresponding BLMR notion, UP-IND-BI.

Given that bidirectional, ciphertext-independent schemes have certain advantages in terms of performance and deployment simplicity, practitioners may prefer them in some cases. For that flexibility, one trades off control over the specificity of rekey tokens, which could be dangerous to confidentiality in some compromise scenarios.

4.4.2 Ciphertext Integrity

We now turn to a notion of integrity captured by the game UP-INT shown in Figure 4.2. The adversary's goal is to submit a ciphertext to the Try oracle that decrypts properly. Of course, we must exclude the adversary from simply resubmitting valid ciphertexts produced by the encryption oracle, or derived from such an encryption by way of re-encryption queries or rekey tokens.

In a bit more detail, in the Try oracle, we define a predicate $\mathsf{InvalidCTXT}$ which captures whether the adversary has produced a trivial derivation of a ciphertext obtained from the encryption oracle. This fulfils a similar role to that of the $\mathsf{Invalid}_{\mathsf{RE}}$ and $\mathsf{Invalid}_{\mathsf{RK}}$ subroutines in the UP-IND game.

For the unidirectional security game UP-INT, we define $\mathsf{InvalidCTXT}(i, C = (\tilde{C}, \overline{C}))$ inductively, outputting true if any of the following conditions hold:

- $i > t$, i.e. k_i is known to the adversary
- $(\tilde{C}, \overline{C})$ was output in response to a query $\mathrm{Enc}(i, m)$
- $(\tilde{C}, \overline{C})$ was output in response to a query $\mathrm{ReEnc}(j, i, C')$ and furthermore $\mathsf{InvalidCTXT}(j, C') = \mathsf{true}$
- $(\tilde{C}, \overline{C})$ is the ciphertext output by running $\mathsf{ReEnc}(\Delta_{j,i,\tilde{C}'}, C')$ for $C' = (\tilde{C}', \overline{C}')$ where $\Delta_{j,i,\tilde{C}'}$ was the result of a query $\mathrm{ReKeyGen}(j, i, \tilde{C}')$ and furthermore $\mathsf{InvalidCTXT}(j, C')) = \mathsf{true}$.

This predicate requires the transcript of queries thus far; to avoid clutter we leave the required transcript implicit in our notation. The definition of $\mathsf{InvalidCTXT}$ is quite permissive: it defines invalid ciphertexts as narrowly as possible, making our security notion stronger. Notably, the adversary can produce any ciphertext (valid or otherwise) using a corrupted key k_i, and use the ReKeyGen oracle to learn a token to update this ciphertext to a non-compromised key. Only the direct re-encryption of the submitted ciphertext is forbidden.

We associate to an updatable encryption scheme Π, an UP-INT adversary $\mathcal{A}$, and parameters κ, t the advantage measure:

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) = \Pr\left[\text{UP-INT}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] \ .$$

4.5 Practical Updatable AE Schemes

We first investigate the security of updatable AE schemes built using the KEM/DEM approach sketched in the introduction and Section 4.3. Such schemes are in widespread use at present, for example in AWS's and Google's cloud storage systems [18, 112], yet have received no formal analysis to date. We produce the AE-hybrid construction as a formalism of this common practice.

Using the confidentiality and integrity definitions from the previous section, we discover that this construction offers very weak security against an adversary capable of compromising keys. Indeed, we are only able to prove security when the number of compromised keys κ is equal to 0. Given the intention of key rotation this is a somewhat troubling result.

On a positive note, we show a couple of simple tweaks to the AE-hybrid which fix these issues. The resultant scheme, named KSS, offers improved security at little additional cost.

4.5.1 Authenticated Encryption

In the following constructions we make use of authenticated encryption (AE) schemes which we define here.

Definition 16 (Authenticated encryption)**.** An authenticated encryption scheme π is a tuple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. $\mathcal{K}$ is a randomised algorithm outputting keys. We denote by $\mathcal{E}_k(\cdot)$ the randomised algorithm for encryption by key k and by $\mathcal{D}_k(\cdot)$ decryption. Decryption is a deterministic algorithm and outputs the distinguished symbol $\perp$ to denote a failed decryption.

In keeping with our definitional choices for updatable AE, we consider randomised AE schemes rather than nonce-based ones.

We use the all-in-one authenticated encryption security definition from [171].

Definition 17 (Authenticated Encryption Security)**.** Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme. Let Enc, Dec be oracles whose behaviours depends on hidden values $b \in \{0, 1\}$ and key $k \leftarrow\$ \mathcal{K}$. Enc takes as input a bit string m and produces $\mathcal{E}_k(m)$ when $b = 0$, and produces a random string of the same length otherwise. Dec takes as input a bit string C and produces $\mathcal{D}_k(C)$ when $b = 0$, and produces $\perp$ otherwise.

Let AE-ROR$_\pi^\mathcal{A}$ be the game in which an adversary $\mathcal{A}$ has access to the Enc and Dec oracles and must output a bit b'. The game outputs true when $b = b'$. We require that the adversary not submit outputs from the Enc oracle to the Dec oracle.

We define the advantage of $\mathcal{A}$ in the AE-ROR security game for π as:

$$\mathsf{Adv}_\pi^{\mathrm{ae}}(\mathcal{A}) = 2 \cdot \Pr\left[\text{AE-ROR}_\pi^\mathcal{A} \Rightarrow \mathsf{true}\right] - 1.$$

Unless otherwise stated, our AE schemes will be length-regular, so that the lengths of ciphertexts depend only on the lengths of plaintexts. This ensures that the above definition also implies a standard “left-or-right” security definition.

4.5.2 (In-)Security of AE-hybrid Construction

Figure 4.3 defines an updatable AE scheme, AE-hybrid, for any AE scheme $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. This is a natural key-wrapping scheme that one might create in the absence of security definitions. It is preferred by practitioners because key rotation is straightforward and performant. Using this scheme means re-keying requires constant time and communication, independent of the length of the plaintext. In fact, we note that this scheme sees widening deployment for encrypted cloud storage services. Both Amazon Web Services [18] and Google Cloud Platform [112] use AE-hybrid to perform key rotations over encrypted customer data.

```
Enc(k, m)                      ReKeyGen(k_1, k_2, $\tilde{C}$)          Dec(k, ($\tilde{C}$, $\overline{C}$))

$x \leftarrow_\$ \mathcal{K}$                   $x = \mathcal{D}(k_1, \tilde{C})$                 $x = \mathcal{D}(k, \tilde{C})$
$\tilde{C} \leftarrow_\$ \mathcal{E}(k, x)$              if $x = \perp$ return $\perp$              if $x = \perp$ return $\perp$
$\overline{C} \leftarrow_\$ \mathcal{E}(x, m)$              $\Delta_{1,2,\tilde{C}} \leftarrow_\$ \mathcal{E}(k_2, x)$          $m = \mathcal{D}(x, \overline{C})$
return $(\tilde{C}, \overline{C})$            return $\Delta_{1,2,\tilde{C}}$                return $m$

KeyGen : return $k \leftarrow_\$ \mathcal{K}$
ReEnc($\Delta_{1,2,\tilde{C}}, (\tilde{C}, \overline{C})$) : return $C' = (\Delta_{1,2,\tilde{C}}, \overline{C})$
```

Figure 4.3: Algorithms for the AE-hybrid updatable AE scheme.

We demonstrate severe limits of AE-hybrid: when keys are compromised confidentiality and integrity cannot be recovered through re-encryption. Later we will demonstrate straightforward modifications to AE-hybrid that allow it to recover both confidentiality and integrity without impacting performance.

Theorem 6 (UP-IND Insecurity of AE-hybrid). *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and Π be the updatable AE scheme AE-hybrid using π as defined in Figure 4.3.*

Then there exists an adversary $\mathcal{A}$ making two queries such that $\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) = 1$ for all $\kappa \geq 1$ and $t \geq 1$.

Proof. We construct a concrete adversary $\mathcal{A}$ satisfying the theorem statement.

$\mathcal{A}$ makes an initial query to $\text{LR}(1, m_0, m_1)$ for distinct messages $m_0 \neq m_1$ and receives challenge ciphertext $C^* = (\mathcal{E}(k_1, x), \mathcal{E}(x, m_b))$. $\mathcal{A}$ subsequently queries the re-encryption oracle for $\text{ReEnc}(1, t+1, C^*)$. k_{t+1} is corrupted and thus $\mathsf{Invalid}_{\mathsf{RK}}$ returns true, so the adversary receives the re-encrypted ciphertext header $\tilde{C}' = \mathcal{E}(k_{t+1}, x)$.

The adversary decrypts $x = \mathcal{D}(k_{t_1}, \tilde{C}')$, computes $m_b = \mathcal{D}(x, \overline{C}^*)$ and checks whether $m_b = m_0$ or m_1. □

The best one can achieve with this scheme is to prove security when $\kappa = 0$, that is, security is not degraded beyond the underlying AE scheme when the adversary does not obtain any compromised keys. However, such a weak security notion is not particularly interesting, since the intention of key rotation is to recover security in the face of key compromises.

Theorem 7 (UP-IND security of AE-hybrid with no key compromise)**.** *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and Π be the updatable AE scheme AE-hybrid using π as defined above. Then for any adversary $\mathcal{A}$ for the game* UP-IND*, there exists an adversary $\mathcal{B}$ for the AE security game where:*

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,0,t}(\mathcal{A}) \;\leq\; 2(t+1) \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B})$$

for all $t \geq 1$.

Proof. We argue using a series of games that the advantage of any adversary in the UP-IND game for AE-hybrid is bounded by the advantage in the AE security game for the underlying AE scheme π. The first game, G_0, is the UP-IND game for AE-hybrid using the underlying scheme π. For $1 \leq i \leq t$, in game G_i we modify the LR oracle by dropping one key k_i, and instead replacing $\tilde{C}$ with a random string of the same length (and store the $\tilde{C}$ and x value) used to answer the query. The ReKeyGen and ReEnc oracles use these stored values to respond.

Let S_i be the event that G_i outputs true. We claim that for $0 < i \leq t$, there exists $\mathcal{B}$ such that:

$$|\Pr[S_{i-1}] - \Pr[S_i]| \leq \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}).$$

We construct this $\mathcal{B}$ by replacing the random string selection for oracle queries with key index i with calls to the Enc oracle in the AE security game for π. When the hidden bit $\widehat{b}$ in the AE game is 0, then $\mathcal{B}$ perfectly simulates game G_{i-1} for key k_i and when $\widehat{b} = 1$, then $\mathcal{B}$ perfectly simulates game G_i.

Similarly, in game G_{t+1} the ciphertext body $\overline{C}$ is replaced by a random string of the same length and again we have:

$$|\Pr[S_t] - \Pr[S_{t+1}]| \leq \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}).$$

And we construct $\mathcal{B}$ in the same way as before, replacing $\overline{C}$ in the LR oracle with a call to the AE encryption oracle. Depending on the value of the hidden bit in the AE security game, either G_t or G_{t+1} is perfectly simulated.

Combining our two claims above we see that:

$$|\Pr[S_0] - \Pr[S_{t+1}]| \leq (t+1) \cdot \mathsf{Adv}^{\mathrm{ae}}_{\pi}(\mathcal{B}).$$

Finally, observe that the adversary in game G_{t+1} learns nothing about the encrypted message m_b, since it is given only random strings. Hence $\Pr[S_{t+1}] = \frac{1}{2}$ and we conclude:

$$\begin{aligned} \mathsf{Adv}^{\text{up-ind}}_{\Pi,0,t}(\mathcal{A}) &= 2 \cdot \Pr[S_0] - 1 \\ &= 2 \cdot (\Pr[S_0] - \Pr[S_{t+1}] + \Pr[S_{t+1}]) - 1 \\ &\leq 2(t+1) \cdot \mathsf{Adv}^{\mathrm{ae}}_{\pi}(\mathcal{B})\ . \end{aligned}$$

□

Unsurprisingly, AE-hybrid is also trivially insecure in the UP-INT sense when $\kappa \geq 1$.

Theorem 8 (UP-INT insecurity of AE-hybrid)**.** *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and Π be the updatable AE scheme AE-hybrid using π as defined in Figure 4.3.*

Then there exists an adversary $\mathcal{A}$ making two queries and one Try *query such that* $\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) = 1$ *for all $\kappa \geq 1$ and $t \geq 1$.*

Proof. We construct a concrete adversary $\mathcal{A}$ satisfying the theorem statement.

$\mathcal{A}$ first queries $\mathrm{Enc}(1, m)$ to obtain an encryption $C = (\mathcal{E}(k_1, x), \mathcal{E}(x, m))$, and subsequently queries $\mathrm{ReEnc}(1, t+1, C)$, receiving the re-encrypted ciphertext $C' = (\mathcal{E}(k_{t+1}, x), \mathcal{E}(x, m))$. Since $\mathcal{A}$ has key k_{t+1}, $\mathcal{A}$ recovers $x = \mathcal{D}(k_{t+1}, \tilde{C}')$ by performing the decryption locally.

Finally, $\mathcal{A}$ constructs the ciphertext $C^* = (\tilde{C}, \mathcal{E}(x, m'))$ for some $m' \neq m$ and queries $\mathrm{Try}(1, C^*)$. Since C^* is not derived from C and k_1 is not compromised, UP-INT outputs true. □

As before, we can only prove UP-INT security of AE-hybrid when $\kappa = 0$.

Theorem 9 (UP-INT Security of AE-hybrid with no key compromise). *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and Π be the AE-hybrid scheme using π as defined above. Then for any adversary $\mathcal{A}$ for the game* UP-INT *there exists an adversary $\mathcal{B}$ for the AE security game where:*

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,0,t}(\mathcal{A}) \;\leq\; (t+1) \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B})$$

for all $t \geq 1$.

Proof. We use the same sequence of games as in the previous proof, leveraging the integrity properties of authenticated encryption.

In game G_i for $1 \leq i \leq t$, for any query (j, C) to the Try oracle where $j \leq i$, the decryption of the ciphertext header $\tilde{C}$ will always decrypt to $\perp$, unless $\tilde{C}$ was a previously generated random string output by the encryption oracle.

Hence in game G_t, the adversary can only win by re-using a previously seen header $\tilde{C}$ which is associated to a DEK x, and constructing a distinct message body $\overline{C}'$.

Between games G_t and G_{t+1}, we again replace the ciphertext body by a call to the AE encryption oracle. Game G_t corresponding to the real world, and G_{t+1} to the random world.

As stated previously, the adversary can only win (i.e. produce a distinct, unique ciphertext) by submitting a distinct, valid ciphertext body. However, in G_{t+1}, the decryption oracle always returns $\perp$ and hence the adversary wins with probability 0. Finally, we conclude:

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,0,t}(\mathcal{A}) \;\leq\; (t+1) \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B})\,.$$

□

```
$\mathsf{Enc}(k, m)$
    $x, y \leftarrow_{\$} \mathcal{K}$
    $r \leftarrow_{\$} \{0,1\}^{\lambda}$
    $\chi = x \oplus y$
    $\overline{C}^2 \leftarrow_{\$} \mathcal{E}(x, m)$
    $\tau \leftarrow_{\$} h(r \parallel x \parallel h(\overline{C}^2))$
    $\tilde{C} \leftarrow_{\$} \mathcal{E}(k, \chi \parallel \tau)$
    return $(\tilde{C}, (r, y, \overline{C}^2))$
```

```
$\mathsf{ReKeyGen}(k_1, k_2, \tilde{C})$
    $(\chi \parallel \tau) = \mathcal{D}(k, \tilde{C})$
    $y' \leftarrow_{\$} \mathcal{K}$
    return $(y', \mathcal{E}(k_2, \chi \oplus y' \parallel \tau))$
```

```
$\mathsf{Dec}(k, (\tilde{C}, \overline{C}))$
    $(\chi \parallel \tau) = \mathcal{D}(k, \tilde{C})$
    if $(\chi \parallel \tau) = \perp$
        return $\perp$
    $r = \overline{C}^0$
    $x = \chi \oplus \overline{C}^1$
    if $\tau \neq h(r \parallel x \parallel h(\overline{C}^2))$
        return $\perp$
    $m = \mathcal{D}(x, \overline{C}^2)$
    return $m$
```

```
$\mathsf{KeyGen}()$ : return $k \leftarrow \mathcal{K}$
$\mathsf{ReEnc}(\Delta_{1,2,\tilde{C}}, (\tilde{C}, \overline{C}))$ : return $(\Delta^1_{1,2,\tilde{C}}, (\overline{C}^0, \overline{C}^1 \oplus \Delta^0_{1,2,\tilde{C}}, \overline{C}^2))$
```

Figure 4.4: Algorithms for the KSS updatable AE scheme.

4.5.3 Improving AE-hybrid

We make small modifications to AE-hybrid and show that the resulting construction has both UP-IND and UP-INT security. These modifications include masking the DEM key stored inside the ciphertext header (to gain UP-IND security), and including an encrypted hash of the message (for UP-INT). We note that these modifications are straightforward to implement on top of the AE-hybrid scheme and have only minimal impact on the scheme's performance.

Let $(\mathcal{K},\mathcal{E},\mathcal{D})$ be an AE scheme and h a hash function with ℓ_h output bits. Then we define KSS (KEM/DEM with Secret Sharing) in Figure 4.4.

The motivation behind KSS is as follows:

First, apply a secret sharing scheme to the DEM key, in this case simply implemented using XOR as $x \mapsto (x \oplus y, y)$. One half is stored encrypted inside the ciphertext header, and the other share is stored with the ciphertext body.

On each re-keygen/re-encryption update the secret shares by generating a random value y' and computing $(x \oplus y \oplus y', y \oplus y')$. Note that knowing two shares from *different* epochs is not sufficient to recover the secret.

This is sufficient to achieve UP-IND security, however, achieving UP-INT security is a more subtle problem. Although using secret sharing to split the DEM key allows the UP-IND proof to go through, it also introduces malleability of the DEM key. This malleability undermines the generic AE properties. As an illustration of this issue, suppose the AE scheme produces keys which have an additional redundant bit. The adversary can easily flip a bit of the ciphertext value y, leaving the decryption unaffected. Similar issues arise whenever attempting to rely on the integrity of the ciphertext encryption in other schemes. Furthermore, in UP-INT, the adversary is able to learn the DEM key x.

Our solution is to bind the ciphertext body to the ciphertext header by storing commitments to various values. In the definition of KSS, we have simply used the ROM commitment scheme $x \mapsto (r, h(r\|x))$. In KSS, we use this scheme to commit to the values of the DEM key x, and the ciphertext body $\overline{C}^2$.

Theorem 10 (UP-IND Security of KSS)**.** *Let* $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a symmetric encryption scheme and* Π *be the updatable AE scheme* KSS *using* π *as defined in Figure 4.4. Then for any adversary* $\mathcal{A}$ *for the game* UP-IND*, making at most* q *queries to the* LR *oracle, there exists an adversary* $\mathcal{B}$ *for the AE security game where:*

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) \ \leq \ 2(t+q) \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B})$$

for all $\kappa \geq 0, t \geq 1$.

Proof. We argue using a series of games that the advantage of any adversary in the UP-IND game for KSS is bounded by the advantage in the AE security game for the underlying AE scheme π. The first game, G_0, is the UP-IND game for KSS using the underlying scheme π. For $1 \leq i \leq t$, in game G_i we replace all ciphertext headers encrypted under key k_i and instead return a random string of the same length used to answer the query. The value of the returned ciphertext header $\tilde{C}$ is stored along with the encrypted value $(\chi \parallel \tau)$ in order to simulate later calls to ReEnc and ReKeyGen.

Let S_i be the event that G_i outputs true. We claim that for $0 < i \leq t$, there exists $\mathcal{B}$ such that:

$$|\Pr[S_{i-1}] - \Pr[S_i]| \leq \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}).$$

We construct $\mathcal{B}$ by using the Enc oracle in the AE security game for π to encrypt ciphertext headers. When the hidden bit $\widehat{b}$ in the AE game is 0, then $\mathcal{B}$ perfectly simulates game G_{i-1} for key k_i and when $\widehat{b} = 1$, then $\mathcal{B}$ perfectly simulates game G_i. $\mathcal{B}$ simply returns as a guess $\widehat{b} = 1$ if the adversary is correct. Any difference between the success probabilities in G_{i-1} and G_i results in an advantage for the adversary.

Now, we observe that the adversary in game G_t cannot learn anything about the DEM key x used to encrypt a challenge. Even through a re-encryption to a corrupted key, the most the adversary can learn is the value $\chi' = x \oplus y'$, where y' is in the ciphertext body and unobtainable by the adversary.

Hence consider another set of hybrids G_j for $t < j \leq t{+}q$, where the adversary makes at most q queries to the left-or-right oracle LR. In the same spirit as the previous hybrids, we construct an AE adversary such that $|\Pr[S_{j-1}] - \Pr[S_j]| \leq \mathsf{Adv}_{\pi}^{\mathrm{ae}}(\mathcal{B})$, this time replacing the encryption of the DEM during a challenge query with the output of the AE encryption oracle.

Finally, observe that in game G_{t+q}, all outputs from LR oracle are of the form $(\tilde{C}, r, y, z)$ where $\tilde{C}, r, y, z$ are all random strings, revealing nothing about the input message. Hence the adversary can learn nothing from oracle queries and so $\Pr[S_{t+q}] = \frac{1}{2}$ and we conclude:

$$\begin{aligned}\mathsf{Adv}_{\Pi,\kappa,t}^{\text{up-ind}}(\mathcal{A}) &= 2 \cdot \Pr[S_0] - 1 \\ &= 2 \cdot (\Pr[S_0] - \Pr[S_1] + \Pr[S_1] + \cdots - \Pr[S_{t+q}] + \Pr[S_{t+q}]) - 1 \\ &\leq 2(t+q) \cdot \mathsf{Adv}_{\pi}^{\mathrm{ae}}(\mathcal{B}) \ .\end{aligned}$$

□

As we will see in the following theorem, collision resistance of the hash function is sufficient to provide UP-INT security, since the hash is integrity-protected by the AE encryption of the KEM. The hash is encrypted to avoid compromise of the ciphertext header being sufficient to distinguish messages.

We achieve collision resistance by assuming h to be a random oracle. However, this assumption could be avoided by either re-using the DEM key x to additionally key the hash function.

Theorem 11 (UP-INT Security of KSS)**.** *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, h be a cryptographic hash function modelled as a random oracle with output length ℓ_h, and Π be the updatable AE scheme* KSS *using π and h as defined in Figure 4.4. Then for any polynomial-time adversary $\mathcal{A}$, making at most q_h queries to the random oracle h, there exists an adversary $\mathcal{B}$ for the AE security game where:*

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) \ \leq \ t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + \frac{q_h^2}{2^{\ell_h}},$$

for all $\kappa \geq 0, t \geq 1$.

Proof. We use the same sequence of games as in the previous proof, leveraging the integrity properties of authenticated encryption.

In game G_i for $1 \leq i \leq t$, for any query (j, C) to the Try oracle where $j \leq i$, the decryption of the ciphertext header $\tilde{C}$ will always decrypt to $\perp$, unless $\tilde{C}$ was a previously generated random string output by the encryption oracle.

Hence in game G_t, the adversary can only win by re-using a previously seen header $\tilde{C}$.

Therefore, the adversary can only win by submitting a ciphertext of the form: $(\tilde{C}, \overline{C}')$, where $\overline{C}' \neq \overline{C}$, the ciphertext body returned in the same query as $\tilde{C}$.

Let $\overline{C} = (r, y, \overline{C}^2)$ and thus we require $\overline{C}' = (r', y', \mathcal{E}(\chi \oplus y', m'))$ for any of $r' \neq r$, $y' \neq y$ or $\mathcal{E}(\chi \oplus y', m') \neq \overline{C}^2$. Note that the adversary can learn the values of x, y used in $\overline{C}$ by querying ReKeyGen$(i, t+1, \tilde{C})$ and receiving $\mathcal{E}(k_{t+1}, \chi \parallel \tau), (r, y, \mathcal{E}(x, m))$. From this, they can simply decrypt and learn $x = \chi \oplus y$

First of all, the ciphertext header contains a commitment to the values x and $h(\overline{C}^2)$. The implication is that the adversary should be unable to modify either x or $\overline{C}^2$ without breaking the commitment scheme. Since KSS uses a simple random oracle commitment scheme, we directly reduce security to the probability of finding a collision in h.

The probability that the adversary finds inputs $r', x', \overline{C}'^2$ such that $h(r' \| x' \| h(\overline{C}'^2)) = h(r \| x \| h(\overline{C}^2))$ and at least one value is distinct, is given by $\frac{q_h^2}{2^{\ell_h}}$. Note that this captures the adversary finding either a collision $h(\overline{C}^2) = h(\overline{C}'^2)$, or of the entire commitment.

Otherwise, the adversary must leave the values $r, x, \overline{C}^2$ unchanged. x is derived by computing $x = \chi \oplus y$, where χ is stored in the ciphertext header and, as established earlier, is unchanged. Hence, the adversary cannot modify y without modifying x.

Therefore, we see that the adversary can only win when finding a collision in h, and so the probability that the adversary wins in G_{t+1} is bounded by this, and we conclude that:

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) \;\leq\; t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + \frac{q_h^2}{2^{\ell_h}}\,.$$

□

One interesting aspect of this proof is that we do not rely on the non- malleability of Π over the ciphertext body for security. This is achieved through the AE-encryption of the commitment of the ciphertext. Hence, KSS could be relaxed slightly by allowing any IND-CPA scheme for the ciphertext body.

4.6 XOR-KEM: A Bidirectional Updatable AE Scheme

The AE-hybrid and KSS schemes are unidirectional and ciphertext-dependent. This means that in practice the client must fetch, from storage, ciphertext headers prior to computing rekey tokens for updating ciphertexts. It would be simpler to utilise a ciphertext-independent scheme that has rekey tokens that work for any ciphertext encrypted with a particular key. This would make the re-encryption process "non-interactive", requiring that the key holder only push a single rekey token to the place where ciphertexts are stored. Given the obvious performance benefits that such a scheme would have, we also provide such a scheme, called XOR-KEM. This scheme is exceptionally fast, and is built from a (non-updatable) AE scheme that is assumed to be secure against a restricted form of related-key attack (RKA). This latter notion adapts the Bellare-Kohno RKA-security notions for block ciphers [30] to the setting of AE schemes. To the best of our knowledge, this definition is novel, and RKA secure AE may itself be of independent interest as a primitive. However, the XOR-KEM scheme cannot meet our integrity notions against an attacker in possession of compromised keys. (And because of its bidirectionality, XOR-KEM also provides the counter-example that we used to separate UP-IND-BI and UP-IND security in Section 4.4.1.)

Let ($\mathcal{K}$,$\mathcal{E}$,$\mathcal{D}$) be an AE scheme. Then we define the ciphertext-independent scheme, XOR-KEM, as follows:

- $\mathsf{KeyGen}()$: **return** $k \leftarrow \mathcal{K}$
- $\mathsf{Enc}(k, m)$: $x \leftarrow \mathcal{K}$; $C \leftarrow (x \oplus k, \mathcal{E}(x, M))$; **return** C
- $\mathsf{ReKeyGen}(k_1, k_2)$: **return** $\Delta_{1,2} = k_1 \oplus k_2$
- $\mathsf{ReEnc}(\Delta_{1,2}, C = (C_0, C_1))$: $C' \leftarrow (\Delta_{1,2} \oplus C_0, C_1)$; **return** C'
- $\mathsf{Dec}(k, C = (C_0, C_1))$: **return** $\mathcal{D}(C_0 \oplus k, C_1)$

The XOR-KEM scheme has a similar format to the AE-hybrid scheme above. However, instead of protecting the DEM key x by encrypting it, we instead XOR it with the secret key k. The resulting scheme becomes a bidirectional, ciphertext-independent scheme, and one that has extremely high performance and deployability.

Note that although the value $x \oplus k$ fulfils a similar purpose as the ciphertext header in AE-hybrid, since this value is not needed in re-keying, it resides in the ciphertext body.

Theorem 12 (UP-IND-BI Security of XOR-KEM)**.** *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and let Π be the updatable AE scheme XOR-KEM built using π as above. Then for any* UP-IND-BI *adversary $\mathcal{A}$ against Π, there exists an AE adversary $\mathcal{B}$ against π such that:*

$$\mathsf{Adv}^{\text{up-ind-bi}}_{\Pi,\kappa,t}(\mathcal{A}) \;\leq\; 2 \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B})$$

for all $\kappa \geq 0, t \geq 1$.

Proof. We consider a sequence of games $G_0, \ldots, G_{q+1}$, where q is the number of queries made by $\mathcal{A}$ to its LR oracle.

Let game G_0 correspond to the regular UP-IND-BI game. Let S_i correspond to the event that game G_i outputs true.

In game G_1, a key $x^* \leftarrow\!\!\$\, \mathcal{K}$ is generated at the start of the game. This is used to encrypt the message in the first query made to the LR oracle. That is, on input (i, m_0, m_1), the LR oracle computes:

$$x \leftarrow\!\!\$\, \mathcal{K};\ \overline{C} \leftarrow\!\!\$\, (x \oplus k_i, \mathcal{E}(x^*, m_b))\,,$$

and returns $\overline{C}$.

All other queries are answered as in G_0. The adversary has the same view in both games, unless the adversary recovers k_i (in which case the adversary can win with one extra query). Either way, we have $\Pr[S_0] = \Pr[S_1]$.

For $1 < \tau \leq q$, game G_τ is identical to $G_{\tau-1}$ except for the adversary's τ-th LR query. There, encryption is computed as:

$$x_\tau \leftarrow\!\!\$\, \mathcal{K};\ \overline{C} \leftarrow\!\!\$\, (x_\tau \oplus k_i, \mathcal{E}(x^*, m_b)),$$

where x^* is the same key generated at the start of G_1. As before, these games are identical to the adversary, and therefore $\Pr[S_\tau] = \Pr[S_{\tau-1}]$.

In game G_{q+1}, we replace encryption by x^* with randomly sampled values. It is straightforward to construct an adversary $\mathcal{B}$ in the AE-ROR game such that $|\Pr[S_{q+1}] - \Pr[S_q]| \leq \mathsf{Adv}^{\mathrm{ae}}_{\pi}(\mathcal{B})$.

Finally, in G_{q+1} the adversary can learn nothing about which m_b is encrypted, therefore $\Pr[S_{q+1}] = \frac{1}{2}$. Combining the above, we get the stated result. $\square$

XOR-KEM does not provide integrity guarantees in the face of compromised keys: an attacker who learns both $C = \mathsf{Enc}(k_1, m)$ and $C' = \mathsf{ReEnc}(\Delta_{1,t+1,\tilde{C}}, C)$ can derive k_1.

For ciphertext integrity of bidirectional schemes, we modify the $\mathsf{InvalidCTXT}$ conditions to check whether the ciphertext is a re-encryption using a bidirectional rekey token. We call this new predicate $\mathsf{InvalidCTXT}_{\mathsf{BI}}$ which returns true whenever $\mathsf{InvalidCTXT}$ returns true, and additionally returns true if:

- $(\tilde{C}, \overline{C})$ is the ciphertext output by running $\mathsf{ReEnc}(\mathsf{Invert}(\Delta_{j,i}), C')$ for $C' = (\tilde{C}', \overline{C}')$, where $\Delta_{j,i}$ was the result of a query ReKeyGen(j, i), and when it holds that $\mathsf{InvalidCTXT}_{\mathsf{BI}}(i, C')) = \mathsf{true}$.

We refer to the security game using $\mathsf{InvalidCTXT}_{\mathsf{BI}}$ as UP-INT-BI.

Unfortunately, proving that XOR-KEM even achieves UP-INT-BI security for no compromised keys ($\kappa = 0$) is not straightforward. It is clear that the adversary can produce trivial manipulations of the ciphertext header: $\tilde{C} \oplus z = (x \oplus k) \oplus z = (x \oplus z) \oplus k$. Thus, to prove UP-INT-BI security additionally requires us to assume that the AE scheme π used in the construction is secure against related-key attacks, in which the adversary can access the encryption and decryption functions of π under XOR-offsets of the unknown key.

Definition 18 (Related-Key Secure AE)**.** Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme with keyspace $\mathcal{S}_{\mathcal{K}}$ and let Φ be a set of functions $\Phi = \{\phi : \mathcal{S}_{\mathcal{K}} \to \mathcal{S}_{\mathcal{K}}\}$. Let the Φ-restricted RKA security game be the game in which the adversary has access to a pair of oracles $\mathsf{Enc}, \mathsf{Dec}$ which, on input (ϕ, x), $\phi \in \Phi$, return for $b = 0$:

$$\mathsf{Enc}(\phi, x) = \mathcal{E}(\phi(k), x), \ \mathsf{Dec}(\phi, x) = \mathcal{D}(\phi(k), x),$$

and for $b = 1$:

$$\mathsf{Enc}(\phi, x) = \$(\cdot), \ \mathsf{Dec}(\phi, x) = \perp,$$

where $k \leftarrow_{\$} \mathcal{K}$ and $b \leftarrow_{\$} \{0, 1\}$ are sampled at the start of the game.

The $\mathrm{RKA}^{\mathcal{A}}_{\pi,\Phi}$ game for encryption scheme π, family of functions Φ, and adversary $\mathcal{A}$ outputs true if the adversary outputs the correct bit b at the end of the game.

We define the advantage of an adversary $\mathcal{A}$ by:

$$\mathsf{Adv}^{\text{rka-ae}}_{\pi,\Phi}(\mathcal{A}) = 2 \cdot \Pr\left[\text{RKA-AE}^{\mathcal{A}}_{\pi,\Phi} \Rightarrow \mathsf{true}\right] - 1.$$

Theorem 13 (UP-INT-BI Security of XOR-KEM)**.** *Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme with keyspace $\{0, 1\}^n$ and let Π be the updatable AE scheme XOR-KEM built using π as defined above. Furthermore, let Φ be the set of permutations $\Phi := \{XOR_\Delta : \Delta \in \{0, 1\}^n\}$ where $XOR_\Delta : \{0, 1\}^n \to \{0, 1\}^n$ denotes the function $XOR_\Delta(k) = k \oplus \Delta$.*

Then for any adversary $\mathcal{A}$ for the game UP-INT*, there exists an adversary $\mathcal{B}$ for the Φ-restricted RKA security game where:*

$$\mathsf{Adv}^{\text{up-int-bi}}_{\Pi,0,t}(\mathcal{A}) \ \leq \ \mathsf{Adv}^{\text{rka-ae}}_{\pi,\Phi}(\mathcal{B})$$

for all $t \geq 1$.

Proof. Let game G_0 be the original UP-INT-BI game.

Let game G_1 be identical to G_0, except we replace all encryption queries with random strings, and condition on the event that every query (i, C) the adversary makes to the Try oracle does not result in setting the win flag to true.

We show that $|\Pr[S_0] - \Pr[S_1]| \leq \mathsf{Adv}^{\text{rka-ae}}_{\pi,\Phi}(\mathcal{B})$. To see this, we construct an adversary $\mathcal{B}$ which simulates the UP-INT-BI game.

For every encryption query to the underlying encryption scheme $\mathcal{E}$, the adversary instead makes a call to the RKA oracle $\mathsf{Enc}(\text{XOR}_x, m)$, where x is as usual sampled uniformly at random for each ciphertext.

In effect, the key used for encrypting the data is $K + x$ where K is the key randomly generated in the RKA game.

When $b = 0$ in the RKA game, encryption queries are returned as $\mathcal{E}(K + x, m)$. This perfectly simulates G_0.

When $b = 1$, decryption queries are all replaced by $\perp$. Therefore, every (i, C) submitted to the Try oracle was either seen before, and hence $\mathsf{InvalidCTXT}_{\mathsf{BI}}(i, C)$ returns true, or results in a call to $\mathsf{Dec}(\text{XOR}_x(i), C)$ which returns $\perp$.

Therefore this perfectly simulates game G_1.

Finally, $\Pr[S_1] = 0$, and thus we get the stated result. □

We conjecture that an authenticated encryption scheme achieving this new notion could be obtained by using a RKA-secure PRF to build a CTR mode cipher, with the PRF also serving as the MAC in encrypt-then-mac mode.

4.7 Indistinguishability of Re-encryptions

The KSS scheme in Section 4.5.3 achieves message confidentiality and ciphertext integrity, even though the actual DEM key is not modified in the course of performing a rotation. Modifying the scheme to ensure the DEM key is also rotated is non-trivial, requiring either significant communication complexity (linear in the length of the encrypted message) between the key server and storage, or the introduction of more advanced primitives such as key-homomorphic PRFs. The question that arises is whether or not changing DEM keys leaves KSS vulnerable to attacks not captured by the definitions introduced thus far.

BLMR's brief treatment of updatable encryption attempts to speak to this issue by requiring that all randomness be refreshed during a rotation. Intuitively this would seem to improve security, but the goal they formalise for this, detailed below, is effectively a correctness condition (i.e., it does not seem to account for adversarial behaviours). It does not help clarify what attacks would be ruled out by changing DEM keys.

Exfiltration attacks. We identify an issue with our KSS scheme (and the other schemes in the preceding section) in the form of an attack that is not captured by the confidentiality definitions introduced so far.

Consider our simple KSS scheme in the context of our motivating key server and storage service application (described in Section 4.3). Suppose an attacker compromises for some limited time both the key server and the storage service. Then for each ciphertext $(\tilde{C}, \overline{C})$ encrypted under a key k_1, the attacker can compute the DEM key $y \oplus \chi = x$ and exfiltrate it.

Suppose the compromise is cleaned up, and the service immediately generates new keys and rotates all ciphertexts to new secret keys. For the KSS scheme, the resulting ciphertexts will still be later decryptable using the previously exfiltrated DEM keys.

Although a confidentiality issue — the attacker later obtains access to plaintext data they should not have — our UP-IND security notion (and, by implication, the weaker BLMR confidentiality notion) do not capture these attacks. Technically this is because the security game does not allow a challenge ciphertext to be encrypted to a compromised key (or rotated to one). Intuitively, the UP-IND notion gives up on protecting the plaintexts underlying such ciphertexts, as the attacker in the above scenario already had access to the plaintext in the first phase of the attack.

One might therefore argue that this attack is not very important. All of the plaintext data eventually at risk of later decryption was already exposed to the adversary in the first time period because she had access to both the key and ciphertexts. But quantitatively there is a difference: for a given ciphertext an adversary in the first time period can exfiltrate just $|x|$ bits per ciphertext to later recover as much plaintext as she likes, whereas the trivial attack may require exfiltrating the entire plaintext.

The chosen-message attack game of UP-IND does not capture different time periods in which the adversary knows plaintexts in the first time period but "forgets them" in the next. One could explicitly model this, perhaps via a two-stage game with distinct adversaries in each stage, but such games are complex and often difficult to reason about (cf., [165]). We instead develop what we believe is a more intuitive route that asks that the re-encryption of a ciphertext should leak nothing about the *ciphertext* that was re-encrypted. We use an indistinguishability-style definition to model this. The interpretation of our definition is that any information derivable from a ciphertext (and its secret key) before a re-encryption is not helpful in attacking the re-encrypted version.

Re-encryption indistinguishability. We formalise this idea via the game shown in Figure 4.5. The adversary is provided with a left-or-right *re-encryption* oracle, ReLR, instead of the usual left-or-right encryption oracle, in addition to the usual collection of compromised keys, a re-encryption oracle, encryption oracle, and rekey token generation oracle. We assume that the adversary always submits ciphertext pairs such that $|C_0| = |C_1|$.

To avoid trivial wins, the game must disallow the adversary from simply re-encrypting the challenge to a corrupted key. Hence we define a $\mathsf{Derived}_{\mathsf{ReLR}}$ predicate, which is identical to the $\mathsf{Derived}_{\mathsf{LR}}$ predicated defined in Section 4.4 for UP-IND security, except that it uses the ReLR challenge oracle. We give it in full detail in the next definition.

```
UP-REENC
    $b \leftarrow_\$ \{0,1\}$
    $k_1, \ldots, k_{t+\kappa} \leftarrow_\$ \mathsf{KeyGen}()$
    $b' \leftarrow_\$ \mathcal{A}^O(k_{t+1}, \ldots, k_{t+\kappa})$
    return $(b' = b)$

Enc$(i, m)$
    return $\mathsf{Enc}(k_i, m)$

ReKeyGen$(i, j, \tilde{C})$
    if $\mathsf{Invalid}_{\mathsf{RK}}(i, j, \tilde{C})$ then
      return $\perp$
    $\Delta_{i,j,\tilde{C}} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
    return $\Delta_{i,j,\tilde{C}}$

ReEnc$(i, j, (\tilde{C}, \overline{C}))$
    $\Delta_{i,j,\tilde{C}} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
    $C' = (\tilde{C}', \overline{C}') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \overline{C}))$
    if $\mathsf{Invalid}_{\mathsf{RE}}(i, j, \tilde{C})$ then
      return $\tilde{C}'$
    else
      return $C'$

ReLR$(i, j, C_0, C_1)$
    if $j > t$ or $|C_0| \neq |C_1|$ then
      return $\perp$
    for $\beta \in \{0,1\}$ do
        $\Delta_{i,j,\tilde{C}_\beta} \leftarrow_\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C}_\beta)$
        $C'_\beta \leftarrow \mathsf{ReEnc}(\Delta i, j, \tilde{C}_\beta, C_\beta)$
        if $C'_\beta = \perp$ then return $\perp$
    return $C'_b$
```

Figure 4.5: The game used to define re-encryption indistinguishability.

Definition 19 (ReLR-derived headers). We define the function $\mathsf{Derived}_{\mathsf{ReLR}}(i, \tilde{C})$ recursively to output true iff $\tilde{C} \neq \varepsilon$ and any of the following conditions hold:

- $\tilde{C}$ was the ciphertext header output in response to a query ReLR(i, C_0, C_1).
- $\tilde{C}$ was the ciphertext header output in response to a query ReEnc(j, i, C') and $\mathsf{Derived}_{\mathsf{ReLR}}(j, \tilde{C}') = \mathsf{true}$.
- $\tilde{C}$ is the ciphertext header output by running $\mathsf{ReEnc}(\Delta_{j,i,\tilde{C}'}, C')$ where $\Delta_{j,i,\tilde{C}'}$ is the result of a query ReKeyGen(j, i, C') for which $\mathsf{Derived}_{\mathsf{ReLR}}(j, \tilde{C}') = \mathsf{true}$.

Then the subroutines $\mathsf{Invalid}_{\mathsf{RK}}, \mathsf{Invalid}_{\mathsf{RE}}$ used in the UP-REENC game output true if $\mathsf{Derived}_{\mathsf{ReLR}}(i, \tilde{C})$ outputs true and $j > t$. We associate to an updatable encryption scheme Π, UP-REENC adversary $\mathcal{A}$, and parameters κ, t the advantage measure:

$$\mathsf{Adv}^{\text{up-reenc}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-REENC}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1 \ .$$

Informally, an updatable encryption scheme is UP-REENC secure if no adversary can achieve advantage far from zero given reasonable resources (run time, queries, and number of target keys).

```
UP-REENC01_m
    $k_1, k_2 \leftarrow\$ \mathsf{KeyGen}()$
    $C_0 \leftarrow\$ \mathsf{Enc}(k_1, m), C_1 \leftarrow\$ \mathsf{Enc}(k_1, m)$
    $\Delta_{1,2,\tilde{C}_1} \leftarrow\$ \mathsf{ReKeyGen}(k_1, k_2, \tilde{C}_1)$
    $C'_1 \leftarrow\$ \mathsf{ReEnc}(\Delta_{1,2,\tilde{C}_1}, C_1)$
    return $(C_0, C'_1)$

UP-REENC00_m
    $k_1, k_2 \leftarrow\$ \mathsf{KeyGen}()$
    $C_0 \leftarrow\$ \mathsf{Enc}(k_1, m), C_1 \leftarrow\$ \mathsf{Enc}(k_1, m)$
    $\Delta_{1,2,\tilde{C}_0} \leftarrow\$ \mathsf{ReKeyGen}(k_1, k_2, \tilde{C}_0)$
    $C'_0 \leftarrow\$ \mathsf{ReEnc}(\Delta_{1,2,\tilde{C}_0}, C_0)$
    return $(C_0, C'_0)$
```

Figure 4.6: Re-encryption indistinguishability experiments from [61].

Notice that exfiltration attacks as discussed informally above would not apply to a scheme that meets UP-REENC security. Suppose otherwise, that the exfiltration still worked. Then one could build an UP-REENC adversary that worked as follows. It obtains two encryptions of different messages under a compromised key, calculates the DEM key (or whatever other information is useful for later decryption) and then submits the ciphertexts to the ReLR oracle, choosing as target a non-compromised key ($j \leq t$). Upon retrieving the ciphertext, it uses the DEM key to decrypt, and checks which message was encrypted. Of course our notion covers many other kinds of attacks, ruling out even re-encryption that allows a single bit of information to leak.

BLMR re-encryption security. BLMR introduced a security goal that we will call basic re-encryption indistinguishability[1]. In words, it asks that the distribution of a ciphertext and its re-encryption should be identical to the distribution of a ciphertext and a re-encryption of a distinct ciphertext of the same message.

More formally we have a pair of experiments shown in Figure 4.6, each parametrised by a message m. Then BLMR require that for all m and all ciphertext pairs (C, C'):

$$|\Pr[\text{UP-REENC00}_m \Rightarrow (C, C')] - \Pr[\text{UP-REENC01}_m \Rightarrow (C, C')]| = 0,$$

where the probabilities are over the coins used in the experiments.

This goal misses a number of subtleties which are captured by our definition. Our definition permits the adversary, for example, to submit *any* pair of ciphertexts to the

[1]BLMR called this ciphertext independence, but we reserve that terminology for schemes that do not require ciphertexts during token generation as per Section 4.2.

ReLR oracle. This includes ciphertexts which are encryptions of distinct messages, and even maliciously formed ciphertexts which may not even decrypt correctly. It is simple to exhibit a scheme that meets the BLMR notion but trivially is insecure under ours[2].

On the other hand, suppose a distinguisher exists that can with some probability ϵ distinguish between the outputs UP-REENC00$_m$ and UP-REENC01$_m$ for some m. Then there exists an adversary against our UP-REENC notion which achieves advantage ϵ. This can be seen by the following simple argument. The adversary gets $C \leftarrow_{\$} \mathsf{Enc}(1, m), C' \leftarrow_{\$} \mathsf{Enc}(1, m)$ and submits the tuple $(1, 2, C, C')$ to its ReLR oracle and receives a re-encryption of one of the ciphertexts, C^*. The adversary then runs the distinguisher on (C, C^*) and outputs whatever the distinguisher guesses. If the distinguisher is computationally efficient, then so too is the UP-REENC adversary. Thus our UP-REENC notion would be stronger than a computational version of the BLMR notion.

4.8 Revisiting the BLMR Scheme

The fact that the simple KEM/DEM schemes of Section 4.5 fail to meet re-encryption security begs the question of finding new schemes that achieve it, as well as UP-IND and UP-INT security. Our starting point is the BLMR construction of an updatable encryption from key-homomorphic PRFs. Their scheme does not (nor did it attempt to) provide integrity guarantees, and so trivially does not meet UP-INT. But before seeing how to adapt it to become suitable as an updatable AE scheme, including whether it meets our stronger notions of UP-IND and UP-REENC security, we first revisit the claims of UP-IND-BI security from [61].

As mentioned in the introduction, BLMR claim that the scheme can be shown secure, and sketch a proof of UP-IND-BI security. Unfortunately the proof sketch contains a bug, as we explain below. Interestingly, revelation of this bug does not lead to a direct attack on the scheme, and at the same time we could not determine if the

[2]Such a scheme can be constructed by starting with a scheme that satisfies both security notions and adding a "counter" component to ciphertexts that records how many re-encryptions have been performed to obtain that ciphertext; one now exploits the property that any pair of ciphertexts can be input to the ReLR oracle in our UP-REENC game, while only fresh ciphertexts C_0, C_1 are rotated in the BLMR notion.

proof could be easily repaired. Instead we are able to show that a proof is unlikely to exist.

Our main result of this section is the following: giving a proof showing the BLMR UP-IND-BI security would imply the existence of a reduction showing that (standard) IND-CPA security implies circular security [48, 68] for a simple KEM/DEM style symmetric encryption scheme. The latter seems quite unlikely given the known negative results about circular security [3, 76], suggesting that the BLMR scheme is not likely to be provably secure.

First we recall some basic tools that BLMR use to build their scheme.

Definition 20 (Key-homomorphic PRF [61])**.** Consider an efficiently computable function $F : \mathcal{K}\times\mathcal{X} \to \mathcal{Y}$ such that $(\mathcal{K}, \oplus)$ and $(\mathcal{Y}, \otimes)$ are both groups. We say that the F is key-homomorphic if for every $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, $F(k_1, x) \otimes F(k_2, x) = F(k_1 \oplus k_2, x)$.

To define security, let game PRF-ROR1$_F^{\mathcal{A}}$ be the game that selects a key $k \leftarrow\!\!\$\ \mathcal{K}$ and then runs an adversary $\mathcal{A}$ that can adaptively query to an oracle that returns F_k applied to the queried message. The adversary outputs a bit. Let game PRF-ROR0$_F^{\mathcal{A}}$ be the game in which an adversary $\mathcal{A}$ can adaptively query an oracle that returns a random draw from $\mathcal{Y}$. The adversary outputs a bit. We assume that $\mathcal{A}$, in either game, never queries the same value twice to its oracle. We define the PRF-ROR advantage of $\mathcal{A}$ as

$$\mathsf{Adv}_F^{\text{prf-ror}}(\mathcal{A}) = \left|\Pr\left[\text{PRF-ROR1}_F^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\text{PRF-ROR0}_F^{\mathcal{A}} \Rightarrow 1\right]\right| \ .$$

A simple example of a secure key-homomorphic PRF in the ROM is the function $F(k, x) = k \cdot H(x)$ where $\mathcal{Y} = \mathbb{G}$ is an additive group in which the decisional Diffie–Hellman assumption holds. This construction is originally due to Naor, Pinkas, and Reingold [152].

As an application of key-homomorphic PRFs, BLMR proposed the following construction. The construction follows a similar approach to the AE-hybrid scheme, but

```
KeyGen()
    $k \leftarrow\!\!\$\ \mathcal{KG}()$
    return $k$

Enc$(k, m)$
    $x \leftarrow\!\!\$\ \mathcal{K}$
    $\tilde{C} \leftarrow\!\!\$\ \mathcal{E}(k, x)$
    $\overline{C} = (m_1 + F(x, 1), \ldots, m_\ell + F(x, \ell))$
    return $C = (\tilde{C}, \overline{C})$

ReKeyGen$(k_i, k_j, \tilde{C})$
    $x = \mathcal{D}(k_i, \tilde{C})$
    $x' \leftarrow\!\!\$\ \mathcal{K}$
    $\tilde{C}' = \mathcal{E}(k_j, x')$
    $\Delta_{i,j,\tilde{C}} = (\tilde{C}', x' - x)$
    return $\Delta_{i,j,\tilde{C}}$

ReEnc$(\Delta_{i,j,\tilde{C}}, C)$
    $(\tilde{C}, \overline{C}) = C$
    $(\tilde{C}', y) = \Delta_{i,j,\tilde{C}}$
    $\overline{C}' = (\overline{C}_1 + F(y, 1), \ldots, \overline{C}_\ell + F(y, \ell))$
    return $C = (\tilde{C}', \overline{C}')$

Dec$(k, C)$
    $(\tilde{C}, \overline{C}) = C$
    $x = \mathcal{D}(k, \tilde{C})$
    $m = (\overline{C}_1 - F(x, 1), \ldots, \overline{C}_\ell - F(x, \ell))$
    return $m$
```

Figure 4.7: The BLMR scheme.

by using a key-homomorphic PRF in place of regular encryption the data encryption key can also be rotated.

Definition 21 (BLMR scheme)**.** Let π be a symmetric-key IND-CPA encryption scheme $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$. Furthermore, let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF where $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are groups, and where the elements of $\mathcal{X}$ can be represented as integers. The BLMR scheme is the tuple of algorithms (KeyGen, Enc, ReKeyGen, ReEnc, Dec) depicted in Figure 4.7.

Note that encryption in the BLMR scheme is a key wrap followed by CTR mode encryption using the wrapped key x and PRF F. For simplicity of presentation we have assumed that a message m can be represented by a sequence $m_1, \ldots, m_\ell$ of elements of $\mathcal{Y}$.

4.8.1 Negative Result about Provable UP-IND Security of BLMR

BLMR sketch a proof for the security of this construction in the UP-IND-BI model (as we refer to it). However, the proof misses a subtle point: the interaction with the ReKeyGen oracle behaves similarly to a decryption oracle and the informal argument given that the IND-CPA security of the KEM is sufficient to argue security

```
Game 1-circular
$b \leftarrow_\$ \{0,1\}$
$k \leftarrow_\$ \mathcal{KG}()$
if $b = 1$ then
    $C \leftarrow_\$ \mathsf{Enc}(k, k)$
else
    $U \leftarrow_\$ \{0,1\}^n$
    $C \leftarrow_\$ \mathsf{Enc}(k, U)$
$b' \leftarrow_\$ \mathcal{A}(C)$
return $(b = b')$
```

```
$\overline{\mathcal{E}}(k, m)$
return $\mathcal{E}(k, m) \| 0$

$\overline{\mathcal{D}}(k, C \| b)$
if $(b = 0)$ then
    return $\mathcal{D}(k, C)$
else
    return $k \oplus \mathcal{D}(k, C)$
```

Figure 4.8: Left: The 1-circular security game. Right: Definition of $\overline{\mathcal{E}}, \overline{\mathcal{D}}$ used in the proof of Theorem 14.

```
$\mathcal{B}^{\mathrm{LR,ReKeyGen}}$
$U \leftarrow_\$ \{0,1\}^n$
$(\tilde{C} \| 0, \overline{C}) \leftarrow_\$ \mathrm{LR}(1, U, 0^n)$
$(\tilde{C}' \| 0, \overline{C}') \leftarrow_\$ \mathsf{ReKeyGen}(1, 1, \tilde{C} \| 1)$
$b' \leftarrow_\$ \mathcal{A}(\tilde{C}' \| 0, \overline{C} + \overline{C}')$
return $b'$
```

Figure 4.9: Adversary $\mathcal{B}$ for UP-IND using as a subroutine the adversary $\mathcal{A}$ attacking 1-circular security of EncBad.

is wrong. In fact, the BLMR scheme seems unlikely to be provably secure even in our basic security model. To argue this, we show that proving security of the BLMR scheme implies the 1-circular security of a specific KEM/DEM construction. Figure 4.8 depicts the security game capturing a simple form of 1-circular security for an encryption scheme $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$.

While our main result here (Theorem 14), can be stated for the BLMR scheme as described earlier, for the sake of simplicity we instead give the result for the special case of using a simple one-time pad DEM instead of the key-homomorphic PRF. This is a trivial example of what BLMR call a key-homomorphic PRG, and their theorem statement covers this construction as well. We will show that proving security for this special case is already problematic, and this therefore suffices to call into question their (more general) theorem. Thus encryption becomes $\mathsf{Enc}(k, m) = (\mathcal{E}(k, r), r \oplus m)$ where $\mathcal{E}$ is an IND-CPA secure KEM. We assume $|m| = n$. We then have $\mathsf{ReKeyGen}(k_1, k_2, \tilde{C}) = (\mathcal{E}(k_2, r'), r' \oplus \mathcal{D}(k_1, \tilde{C}))$. We have the following theorem:

Theorem 14. *Let $\mathcal{E}$ be an IND-CPA-secure symmetric encryption scheme. If there exists a reduction from* BLMR UP-IND-BI-*security to the IND-CPA security of $\mathcal{E}$, then there exists a reduction that shows that* Enc *is 1-circular secure.*

Proof. We start by introducing a slight variant of $\mathcal{E}$, denoted $\overline{\mathcal{E}}$, shown in Figure 4.8. It adds a bit to the ciphertext[3] that is read during decryption: if the bit is 1 then decryption outputs the secret key XOR'd with the plaintext. Let EncBad be the same as Enc above but using $\overline{\mathcal{E}}$, i.e., $\mathsf{EncBad}(k, m) = \overline{\mathcal{E}}(k, r), r \oplus m$ and $\mathsf{ReKeyGenBad}(k_1, k_2, C) = \overline{\mathcal{E}}(k_2, r'), r' \oplus \overline{\mathcal{D}}(k_1, C)$.

If $\mathcal{E}$ is IND-CPA then $\overline{\mathcal{E}}$ is as well. Thus if $\overline{\mathcal{E}}$ is IND-CPA, then the security claim of BLMR implies that EncBad is UP-IND-BI. We will now show that UP-IND-BI security of EncBad implies the 1-circular security of EncBad. In turn it is easy to see that if EncBad is 1-circular secure then so too is Enc, and, putting it all together, the claim of BLMR implies a proof that IND-CPA of $\mathcal{E}$ gives 1-circular security of Enc.

It remains to show that UP-IND-BI security implies EncBad 1-circular security. Let $\mathcal{A}$ be a 1-circular adversary against EncBad. Then we build an adversary $\mathcal{B}$ against the UP-IND security of EncBad. It is shown in Figure 4.9. The adversary makes an LR query on a uniform message and the message 0^n. If the UP-IND-BI challenge bit is 1 then it gets back a ciphertext $C_1 = (\overline{\mathcal{E}}(k_1, r) \| 0, r \oplus U)$ and if it is 0 then $C_0 = (\overline{\mathcal{E}}(k_1, r) \| 0, r)$. Next it queries ReKeyGen oracle on the first component of the returned ciphertext but with the trailing bit switched to 1. It asks for a rekey token for rotating from k_1 back to k_1. The value returned by this query is equal to $\overline{\mathcal{E}}(k_1, r') \| 0, r' \oplus k_1 \oplus r$. By XOR'ing the second component with the second component returned from the LR query the adversary gets finally a ciphertext that is, in the left world, the encryption of k_1 under itself and, in the right world, the encryption of a uniform point under k_1. Adversary $\mathcal{B}$ runs a 1-circular adversary $\mathcal{A}$ on the final ciphertext and outputs whatever $\mathcal{A}$ outputs. □

The above result uses 1-circular security for simplicity of presentation, but one can generalise the result to longer cycles with additional queries.

[3]Notice that this scheme is not tidy in the sense of [151]. While that does not affect the implications of our analysis — BLMR make no assumptions about tidiness — finding a tidy counter-example is an interesting open question.

The result is relative, only showing that a proof of BLMR's claim implies another reduction between circular security and IND-CPA security for the particular KEM/DEM scheme Enc above. It is possible that this reduction exists, however it seems unlikely. Existing counter-examples show IND-CPA schemes that are not circular-secure [122]. While these counter-examples do not have the same form as the specific scheme under consideration, it may be that one can build a suitable counter-example with additional effort.

4.9 An Updatable AE Scheme with Re-encryption Indistinguishability

We first point out that one can avoid the issues raised in Section 4.8 by replacing the IND-CPA KEM with a proper AE scheme. This does not yet, however, address integrity of the full encryption scheme. To provide integrity overall, we can include a hash of the message in the ciphertext header.

This amended construction — which we refer to as ReCrypt — is detailed in Figure 4.10. It uses an AE scheme $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$, a key-homomorphic PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, a hash function $h : \{0,1\}^* \to \mathcal{Y}$, and a commitment scheme com. Here $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are groups. A message m is assumed to be a sequence $m_1, \ldots, m_\ell$ of elements in $\mathcal{Y}$. When we come to discuss specific instantiations in Section 4.9.2, we will explain how to modify the scheme to handle messages that are binary strings. We also assume that elements of $\mathcal{X}$ can be represented by integers (as in the BLMR scheme).

The motivation behind the construction of ReCrypt is the same as for KSS: the ciphertext header uses AE to guarantee the integrity and confidentiality of its contents, which includes a secret-shared DEM key, and a commitment. The key difference is that the encryption mechanism for the ciphertext body uses key-homomorphic PRFs to allow updating, and the commitment scheme is similarly updatable.

We do not cover commitment schemes in detail, but defer to the lecture notes by Bellare for a modern definition [26].

```
$\mathsf{Enc}(k, m)$
$x, y \leftarrow\!\!\$\, \mathcal{K}$
$\chi = x + y$
for $1 \leq l \leq \ell$
    $\overline{C}_l^2 = m_l + F(x, l)$
$z = h(m)$
$(\tau_0, r_0) \leftarrow\!\!\$\, \mathsf{com}(x)$
$(\tau_1, r_1) \leftarrow\!\!\$\, \mathsf{com}(z)$
$\tilde{C} \leftarrow\!\!\$\, \mathcal{E}(k, \chi \parallel \tau_0 \parallel \tau_1)$
return $(\tilde{C}, ((r_0, r_1), y, \overline{C}^2))$
```

```
$\mathsf{ReKeyGen}(k_1, k_2, \tilde{C})$
$(\chi \parallel \tau_0 \parallel \tau_1) = \mathcal{D}(k_1, \tilde{C})$
$x', y' \leftarrow\!\!\$\, \mathcal{K}$
$\chi' = \chi + x' + y'$
$(\tau'_0, r'_0) \leftarrow\!\!\$\, \mathsf{com}(x')$
$(\tau'_1, r'_1) \leftarrow\!\!\$\, \mathsf{com}(0)$
return $(x', y', r'_0, r'_1, \mathcal{E}(k_2, \chi' \parallel \tau_0 \cdot \tau'_0 \parallel \tau_1 \cdot \tau'_1))$
```

```
$\mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \overline{C}))$
$(x', y', r'_0, r'_1, \tilde{C}') = \Delta_{i,j,\tilde{C}}$
$((r_0, r_1), y, \overline{C}^2) = \overline{C}$
for $1 \leq l \leq \ell$
    $\overline{C}_l'^2 = \overline{C}_l^2 + F(x', l)$
$\overline{C}' = ((r_0 + r'_0, r_1 + r'_1), y + y', \overline{C}'^2)$
return $(\tilde{C}', \overline{C}')$
```

```
$\mathsf{Dec}(k, (\tilde{C}, \overline{C}))$
if $\mathcal{D}(k, \tilde{C}) = \perp$
    return $\perp$
$(\chi \parallel \tau_0 \parallel \tau_1) = \mathcal{D}(k, \tilde{C})$
$((r_0, r_1), y, \overline{C}^2) = \overline{C}$
$x = \chi - y$
for $1 \leq l \leq \ell$
    $m_l = \overline{C}_l^2 - F(x, l)$
$z = h(m)$
if $\mathsf{open}(\tau_0, x, r_0) \neq 1 \vee \mathsf{open}(\tau_1, z, r_1) \neq 1$
    return $\perp$
return $m$
```

```
$\mathsf{KeyGen}()$ : return $k \leftarrow \mathcal{K}$
```

Figure 4.10: The ReCrypt scheme. Note that use of the commitment scheme requires a trusted setup to provide public parameters which we omit for simplicity.

Definition 22 (Commitment Scheme)**.** A commitment scheme $\mathcal{CS} = (\mathcal{P}, \mathsf{com}, \mathsf{open})$ is a tuple of algorithms where $\mathcal{P}$ outputs public parameters p, com takes as input the params p, a source of randomness, and a message, and produces a commitment c and a decommital key (or opening key) d. open takes p, c, m, d, as input and returns 1 if the commitment is valid, or 0 otherwise.

For simplicity, we often leave the public parameters p implicit. We write $(c, d) \leftarrow_{\$} \mathsf{com}(m)$, or $c \leftarrow \mathsf{com}(m; d)$, where the latter represents deterministically computing the commitment for a fixed opening key d.

A commitment scheme is said to be *hiding* if an adversary cannot learn the message m from a commitment $c = \mathsf{com}(\mathsf{p}, m)$. This is formally quantified by an adversary $\mathcal{A}$ playing a left-or-right style game with advantage $\mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{A})$.

A commitment scheme is said to be binding if it is hard to find a collision in commitment values such that $\mathsf{open}(\mathsf{p}, c, m, d) = \mathsf{open}(\mathsf{p}, c, m', d') = 1$ with $m' \neq m$ or $d \neq d'$. This is formally quantified by a collision finding adversary $\mathcal{A}$ with advantage $\mathsf{Adv}^{\text{cs-bind}}_{\mathcal{CS}}(\mathcal{A})$.

For ReCrypt, we additionally require the commitment scheme to be homomorphic. For example, the commitment scheme due to Pedersen [158] has this property: $\mathsf{com}(x; r) \cdot \mathsf{com}(y; s) = g^x h^r \cdot g^y h^s = g^{x+y} h^{r+s} = \mathsf{com}(x + y; r + s)$. Homomorphic commitment schemes are also considered in [82, 110]. Note that [110] used homomorphic commitments in the context of multiparty, verifiable secret-sharing, which is in essence the same composite property we require.

For use of the Pedersen scheme, we would additionally need a trusted setup to provide the values g, h as public parameters. However, note that these are never needed by the server, and thus the client only needs to trust their own generation of these values.

In the remainder of this section we show that the new scheme meets our strongest security notions for updatable encryption. We then assess the viability of using this scheme in practice, discussing how to instantiate F for high performance and reporting on performance of the full scheme.

4.9.1 Security of ReCrypt

We state three security theorems for ReCrypt: UP-IND, UP-INT, and UP-REENC notions. The proof of UP-INT relies on the collision resistance of the hash h, while the other two proofs do not. For simplicity, and because we will later instantiate the PRF F in the Random Oracle Model (ROM), we model h as a random oracle throughout our analysis. This modelling of h could be avoided using the approach of Rogaway [168], since concrete collision-producing adversaries can be be extracted from our proofs. Note also that the *almost* key-homomorphic PRF construction in the standard model presented by BLMR would not achieve UP-REENC since the number of re-encryptions is leaked by the ciphertext, allowing an adversary to distinguish two re-encryptions.

Theorem 15 (UP-IND security of ReCrypt)**.** *Let $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an AE scheme, $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF, $h : \{0,1\}^* \to \mathcal{Y}$ be a hash function, $\mathcal{CS}$ be a homomorphic commitment scheme, and let Π be the* ReCrypt *scheme as depicted in Figure 4.10.*

Then for any adversary $\mathcal{A}$ against Π, making at most q queries to the LR *oracle, there exist adversaries $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ such that*

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) \leq 2t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + 2q \cdot \mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{C}) + \mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{D})$$

for all $\kappa \geq 0, t \geq 1$.

Proof. We split the proof into two parts. The first part uses the AE security of π to show that the value of x used to key the key-homomorphic PRF is hidden from the adversary. The second part uses the fact that F is a PRF to show indistinguishability holds, given that the adversary cannot learn x.

Let G_0 be the original UP-IND game. For $1 \leq i \leq t$, Game G_i is the same as G_{i-1}, except we replace the encryption $\mathcal{E}(k_i, \chi \parallel \mathsf{com}(x; r_0) \parallel \mathsf{com}(h(m), r_1))$ with a random string c and store the tuple (x, y, r_0, r_1, m) as a lookup in the table $\mathbb{C}$ indexed by the random string c. For queries to $\mathsf{ReKeyGen}(i, j, \tilde{C})$, if $\tilde{C} = c$ for some previously returned c, then compute r_0', r_1', x', y' as usual and either:

- if $j \leq i$, return (x', y', r'_0, r'_1, c') for some random c', storing the value $(x + x', y + y', r_0 + r'_0, r_1 + r'_1, m)$, or
- if $j > i$, return $(x', y', r'_0, r'_1, \mathcal{E}(k_j, x{+}y{+}x'{+}y' \parallel \mathsf{com}(x{+}x'; r_0{+}r'_0) \parallel \mathsf{com}(h(m), r_1{+}r'_1)))$.

If $\tilde{C}$ has not been previously seen, return $\perp$. These modifications are shown in Figure 4.11.

Let S_i be the event that $\mathcal{A}$ outputs the correct bit in game G_i. Then we can construct an adversary $\mathcal{B}$ such that $|\Pr[S_i] - \Pr[S_{i-1}]| \leq \mathsf{Adv}^{\mathrm{ae}}_{\pi}(\mathcal{B})$. To see this, notice that by replacing the $\mathcal{E}(k_i, \cdot)$ and $\mathcal{D}(k_i, \cdot)$ functions with calls to an instance of the AE game for Π, in the real world, we get G_{i-1}, and the random world corresponds to G_i.

In game G_t, suppose the attacker queries the LR oracle to receive a challenge ciphertext C. Then the header $\tilde{C}$ will be equal to some random string r, while the body $\overline{C}$ consists of the randomness for the commitment scheme (r_0, r_1), the key mask y, and the message encrypted by the PRF F keyed by some value x. Note that the use of AE forces the adversary to submit only previously seen ciphertext headers to oracles with $i \leq t$.

From re-keygen queries, the attacker can learn the fresh values of r'_0, r'_1, x', y'. However, these are insufficient to learn anything about the value of x used to encrypt the message; the adversary can only learn at most one half of the secret-shared key.

Similarly, re-encryption queries where the target key j is uncompromised, i.e., $j \leq t$, the adversary receives a fresh ciphertext encrypted under x' with no way of learning x'.

On the other hand, if $j > t$, the invalidity condition $\mathsf{Invalid}_{\mathsf{RE}}$ will be true, so the adversary learns the ciphertext header. This is of the form $\mathcal{E}(k_j, \chi \parallel (\tau_0.\tau_1)))$. Since the adversary possesses k_j, they may decrypt the ciphertext header. However, x is still masked by y.

Next, we use the hiding property of the commitment scheme. In game G_{t+1}, we replace all messages used in commitments for queries to uncompromised keys $j \leq t$

with random strings. We use the difference in success probabilities for games G_t and G_{t+1} to construct the distinguished $\mathcal{D}$ against the commitment scheme $\mathcal{CS}$.

From here, we proceed with a further q hybrid game hops, at each hop embedding a PRF challenger.

Suppose the adversary makes q queries to the LR oracle, then we define games $G_{t+2}, \ldots G_{t+q+1}$ such that in game G_{t+u+1}, for $1 \leq u \leq q$, for the adversary's u-th query to the LR oracle, we replace the usage of $F(x, l)$ by a uniformly random value r_l.

This is equivalent to replacing $F(x, l)$ in the LR oracle by the PRF challenge oracle, denoted $f(l)$. Suppose K is the key used by the PRF challenger. When the PRF challenge is from the real world, the computed encryption is of the form $m_l + F(x, l) + f(l) = m_l + F(K + x, l)$.

Since we showed that the value of x is unknown to the adversary for challenge ciphertexts, this change perfectly models the game G_{t+u}.

On the other hand, if f is a random function, then outputs are distributed precisely as the random values r_l chosen beforehand, corresponding to game G_{t+u+1}. Hence, we use any difference in win probabilities between G_{t+u} and G_{t+u+1} to construct a PRF adversary $\mathcal{C}$.

Furthermore, the computation of re-keying tokens and re-encryptions does not remove the PRF mask r from the ciphertext: if j is uncorrupted, then we compute $\overline{C}_l + F(x', l) = m_l + F(x + x' + K, l)$; otherwise, either $\mathsf{Invalid}_{\mathsf{RE}}$ or $\mathsf{Invalid}_{\mathsf{RK}}$ will be false, and the oracle returns $\perp$ for the ciphertext body.

Similarly, the adversary can also obtain the ciphertext header containing an encryption of $x + x' + y + y'$, τ_0, and τ_1. However, we replaced the commitments in game G_{t+1} with random values, thus these do not reveal anything about the message, and neither does the value of χ'.

Hence in game G_{t+q+1}, both the message and the hash are perfectly masked by the PRF values, and the adversary cannot win with a probability greater than $\frac{1}{2}$. Hence, we conclude that:

$$\begin{aligned}\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) &= |2 \cdot \Pr[S_0] - 1| \\ &\leq 2t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + 2q \cdot \mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{C}) + \mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{D})\end{aligned}$$

for all $\kappa \geq 0, t \geq 1$. □

```
$\mathsf{Enc}(i, m)$
$x, y \leftarrow\!\!\$\ \mathcal{K}$
$z = h(m)$
$(\tau_0, r_0) \leftarrow\!\!\$\ \mathsf{com}(x)$
$(\tau_1, r_1) \leftarrow\!\!\$\ \mathsf{com}(z)$
$c \leftarrow\!\!\$\ \{0,1\}^{|\tilde{C}|}$
$\mathbb{C}_i[c] = (x, y, r_0, r_1, m)$
for $1 \leq l \leq \ell$
    $\overline{C}^2_l = m_l + F(x, l)$
return $(c, ((r_0, r_1), y, \overline{C}^2))$

$\mathsf{ReKeyGen}(i, j, \tilde{C})$
if $\mathbb{C}_i[\tilde{C}] = \perp$ return $\perp$
$(x, y, r_0, r_1, m) \leftarrow \mathbb{C}_i[\tilde{C}]$
$x', y' \leftarrow\!\!\$\ \mathcal{K}$
$\chi' = x + y + x' + y'$
if $j \leq i$ then
    $c \leftarrow\!\!\$\ \{0,1\}^{|\tilde{C}|}$
    $\mathbb{C}_j[c] = (x + x', y + y', r_0 + r'_0, r_1 + r'_1, m)$
    return $\Delta_{i,j,\tilde{C}} = (x', y', r'_0, r'_1, c)$else
    $(\tau'_0, r'_0) \leftarrow\!\!\$\ \mathsf{com}(x + x')$
    $(\tau'_1, r'_1) \leftarrow\!\!\$\ \mathsf{com}(h(m))$
    $\tilde{C}' \leftarrow\!\!\$\ \mathcal{E}(k_j, (\chi', \tau))$
    return $(x', y', r'_0 - r_0, r'_1 - r_0, \mathcal{E}(k_2, \chi' \| \tau'_0 \| \tau'_1))$

$\mathsf{Dec}(i, (\tilde{C}, \overline{C}))$
if $\mathbb{C}_i[\tilde{C}] = \perp$ return $\perp$
$(x, y, r_0, r_1, m) \leftarrow \mathbb{C}_i[\tilde{C}]$
$((r'_0, r'_1), y', \overline{C}^2) = \overline{C}$
$x' = x + y - y'$
if $\mathsf{open}(\mathsf{com}(x; r_0), x', r'_0) \neq 1$
    return $\perp$
for $1 \leq l \leq \ell$
    $m'_l = \overline{C}_l - F(x, l)$
if $\mathsf{open}(\mathsf{com}(h(m); r_1), h(m'), r'_1) \neq 1$
    return $\perp$
return $m'$
```

Figure 4.11: The replacement algorithms used in game G_i for the proofs of security for the ReCrypt construction. For the i-th key, encryption/decryption by Π is replaced by random strings r of the same length and a lookup.

Theorem 16 (UP-INT security of ReCrypt)**.** *Let $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an AE scheme, $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a key-homomorphic PRF, h be a cryptographic hash function* *modelled as a random oracle with outputs in $\mathcal{Y}$, $\mathcal{CS}$ be a homomorphic commitment scheme, and let* Π *be the* ReCrypt *scheme as depicted in Figure 4.10.*

Then for any adversary $\mathcal{A}$ against Π*, there exists adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) \leq 2t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + \mathsf{Adv}^{\text{cs-bind}}_{\mathcal{CS}}(\mathcal{C}) + \frac{q_h^2}{|\mathcal{Y}|}$$

for all $\kappa \geq 0, t \geq 1$, where the adversary makes q queries to h.

Proof. Apply the same t transformations shown in Figure 4.11, which substitutes encryption by π with random strings for uncorrupted keys. Decryption is then simulated by using a lookup of previously returned values. Let S_i be the event that $\mathcal{A}$ outputs the correct bit in game G_i.

In game G_{t+1}, we replace the opening of the commitment in Dec verification verify that $x' = x$ and $m' = m$. These two games are identical, unless the adversary submits for verification a ciphertext $C = (\tilde{C}, \overline{C})$ such that $\tilde{C}$ has previously been generated for the tuple $(x, y, r_0, r_1 m)$ but either $y' \neq y$ and therefore $x' = \chi - y' \neq x$ or $y' = y$ but $\overline{C}$, decrypts using x to a message m' such that $m' \neq m$ but $\mathsf{open}(\mathsf{com}(h(m); r_0), h(m'), r_0') = 1$.

First of all, suppose that $h(m') \neq h(m)$. Then we can bound the probability that the adversary constructs such a ciphertext by constructing an adversary $\mathcal{C}$ which creates a colliding commitment pair in the binding game. This probability is quantified by $\mathsf{Adv}^{\text{cs-bind}}_{\mathcal{CS}}(\mathcal{C}$.

Otherwise, we must have that $h(m) = h(m')$, i.e. the adversary finds a collision in h. By modelling h as a random oracle, we can bound the probability that the adversary making q queries to the random oracle, succeeds in finding suitable values of m, and m' by $q^2/|\mathcal{Y}|$. Therefore $|\Pr[S_{t+1}] - \Pr[S_t]| \leq \mathsf{Adv}^{\text{cs-bind}}_{\mathcal{CS}}(\mathcal{C}) + q^2/|\mathcal{Y}|$.

Finally, we need to show that the adversary cannot create ciphertext body $\overline{C}' \neq \overline{C}$ which correctly decrypts given the above modifications. From the commitments, the adversary cannot modify y, since x is recomputed as $x' = \chi - y'$, and the value of x is verified. Furthermore, modifying r_0 or r_1 would similarly invalidate the previous assumption that the adversary does not find colliding commitment values. This leaves the encrypted plaintext $\overline{C}^2 = m_l + F(x, l)$. Since m, x are fixed by the previous game hops, this value is uniquely determined since $\mathcal{Y}$ is a group, and hence $m_l + F(x,l) + \epsilon - F(x,l) = m_l \iff \epsilon = 0$.

Hence we have shown that in game G_{t+1} all of the ciphertext components are non-malleable and we have the probability of the adversary winning in game G_{t+1} is 0.

Therefore, the advantage of $\mathcal{A}$ is:

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) \leq t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + \mathsf{Adv}^{\text{cs-bind}}_{\mathcal{CS}}(\mathcal{C}) + \frac{q^2}{|\mathcal{Y}|} .$$

□

Finally, we prove that ReCrypt meets our re-encryption indistinguishability notion.

Theorem 17 (UP-REENC security of ReCrypt)**.** *Let $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an AE scheme, $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a key-homomorphic PRF, $\mathcal{CS}$ be a homomorphic commitment scheme, and let Π be the* ReCrypt *scheme as depicted in Figure 4.10.*

Then for any adversary $\mathcal{A}$ against Π, there exist adversaries $\mathcal{B}$, $\mathcal{C}$ such that

$$\mathsf{Adv}^{\text{up-reenc}}_{\Pi,\kappa,t}(\mathcal{A}) \leq 2t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + 2q \cdot \mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{C}) + \mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{D})$$

for all $\kappa \geq 0, t \geq 1$.

Proof. Apply the same t steps as used in the previous proof, shown in Figure 4.11. Then we have the ciphertext headers replaced with random strings, and the re-keygen process is simulated by recording inputs previously seen.

Hence, in game G_t, the adversary cannot learn anything about the PRF key x used in the challenge.

In game G_{t+1}, we replace the commitment values in the ciphertext header with random values. As in the UP-IND proof, the hiding property of $\mathcal{CS}$ bounds the difference between the success probabilities of games G_t and G_{t+1} by $\mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{D})$.

As before, we construct a series of q hybrid game hops to embed instances of the PRF game. For an adversary making q calls to the ReLR oracle, and $1 \leq u \leq q$, we define game G_{t+u+1} to replace computation of $F(x + x', l)$ by uniformly random values r_l.

In game G_{t+u}, computation of re-encryption is equivalent to using $F(K + x + x, l)$ from the PRF challenge oracle, and game G_{t+u+1} is equivalent to uniformly random values from $f(l)$. Hence the difference in success probability between G_{t+u} and G_{t+u+1} gives the advantage of an adversary $\mathcal{C}$ in the PRF game.

Finally, in game G_{t+q+1}, the re-encrypted ciphertext is perfectly masked by the random values r_l, both the commitment opening values and the shared DEM key is refreshed with uniformly random values, and hence we conclude that the adversary cannot guess b with any better probability than guessing.

The same computation as before results in:

$$\mathsf{Adv}^{\text{up-reenc}}_{\Pi,\kappa,t}(\mathcal{A}) \leq 2t \cdot \mathsf{Adv}^{\text{ae}}_{\pi}(\mathcal{B}) + 2q \cdot \mathsf{Adv}^{\text{prf-ror}}_{F}(\mathcal{C}) + \mathsf{Adv}^{\text{cs-hide}}_{\mathcal{CS}}(\mathcal{D}) \ .$$

$\square$

4.9.2 Implementation and Performance

We dedicate the remainder of this section to analysis of the viability of ReCrypt for use in practice.

We first must instantiate the key-homomorphic PRF used by ReCrypt. While BLMR suggest using a standard model construction, a more efficient route is to use the classic ROM construction due originally to Naor, Pinkas, and Reingold [152]. Here one uses $F(k, x) = k \cdot H(x)$ where H is modelled as a random oracle $H : \mathcal{X} \to \mathbb{G}$, $\mathcal{X}$ is any suitable set (bit-strings of some fixed length, for example), and $(\mathbb{G}, +)$ is a group in which the decisional Diffie–Hellman (DDH) assumption holds. We will use as group $\mathbb{G}$ of subset of the points on an elliptic curve $E(\mathbb{F}_p)$ defined over a prime field. In our implementation we use the specific curve Curve25519 [36], which has $p = 2^{255} - 19$.

For the commitment scheme, we used Pedersen commitments [158] of the form $\mathsf{com}(x; r) = g^x h^r$, where the public parameters are $\mathsf{p} = (g, h)$. The security of this construction relies on the discrete logarithm problem. It is assumed that the adversary cannot solve $\log_g h$. We also use Curve25519 as the group here, and hence we have $\mathsf{com}(x; r) = xP + rQ$ for base points $\mathsf{p} = (P, Q)$.

We use SHA256 for the hash function h and AES128-GCM for the AE scheme π.

Message encoding. Recall that encryption is done block-wise as $\overline{C}_l = m_l + F(x, l)$, where it was assumed that messages m are already represented as sequences over the group $\mathbb{G}$. To make a practical scheme for encrypting data represented as bitstrings, we therefore require an encoding function $\sigma_m : \{0,1\}^n \to \mathbb{G}$ for some block length n. Padding can be used to ensure messages are of length a multiple of n.

For a suitable message encoding function, we require the function and its inverse to be efficiently computable. In order to avoid side-channel leakage on plaintexts, we require the function to be amenable to constant-time implementation. In addition, security mandates that σ_m^{-1} can be implemented in such a way that it rejects on input any element that it is outside the range of σ_m. For otherwise, σ_m^{-1} may not act as a bijection on its inputs; the existence of two points P, P' on the curve such that $\sigma_m^{-1}(P) = \sigma_m^{-1}(P')$ might then allow an adversary to construct a ciphertext forgery.

We use the Elligator function [37] to realise σ_m. For elliptic curve groups including Curve25519 [36], the Elligator function produces an injective mapping from $\{0, 1, \ldots, \frac{p-1}{2}\}$ to points on the curve. The inverse mapping returns elements of the form $r, p - r$; we obtain a bijection by inverting the map and taking as output the value that is less than $\frac{p-1}{2}$.

Using Elligator, the natural block size for messages would be $\lfloor \log{(p-1)} - 1 \rfloor = 254$ bits, which is not an integral number of bytes. However, splitting bytes would have a deleterious impact on performance, and so we actually work with a 31 byte (248 bit) block size. Hence, on recovering the message using σ_m^{-1}, we must additionally check that the top byte is zero.

Hashing to the curve. In order to instantiate the key-homomorphic PRF, we require a hash function H outputting elements in the group $\mathbb{G}$. Here we are less concerned about side-channel attacks because the inputs to H in our scheme are not secret. The natural way to build H would be to take a regular hash function

$h : \{0,1\}^* \rightarrow \{0,1\}^n$, modelled as a random oracle, and apply any encoding function σ_m, resulting in $H(x) := \sigma_m(h(x))$. However, for the function H to additionally act like a random oracle, we would require at least that σ_m maps uniformly to elements in $\mathbb{G}$, imposing additional requirements on σ_m. We therefore require a second encoding function σ_h for constructing the key-homomorphic PRF, and require that σ_h composed with a random oracle h is indifferentiable [140] from a random oracle.

An in-depth treatment of indifferentiable hashing onto elliptic curves appears in [66]. For simplicity, we opted to use their general approach: $\sigma_h(m) = f(h_1(m)) + h_2(m) \cdot G$. Here h_1 and h_2 are cryptographic hash functions suitable to be modelled as random oracles, instantiated in our implementation by SHA-256 with appropriate domain separation. The function f needs to be what [66] refer to as a weak encoding function. Elligator satisfies the necessary properties, as it is polynomial time computable, the probability of any point being output is either 0 or $2/\#\mathbb{G}$, and it is samplable by simply picking from $\pm f^{-1}(P)$ for any $P \in E(\mathbb{F}_p)$.

Point compression. Curve25519 supports an x-coordinate-only Montgomery ladder for scalar multiplication. Since we want to unambiguously add points rather than just computing scalar multiples, we need to work with both x and y coordinates. Therefore, we used the twisted Edwards form of Curve25519, implemented in [131], which in turn is based on [128]. We also used the latter to guide our Elligator implementation.

We save bandwidth using point compression. When a curve point is serialised, only the y coordinate and the sign of the x coordinate (1-bit) needs to be recorded. Since the y coordinate requires less than the full 32 bytes, we are able to serialise points as 32 byte values. Each 32 byte serialised value represents 31 bytes of plaintext, giving a ciphertext expansion of 3%. Upon deserialisation, the x coordinate must be recomputed. This requires computing a square root, taking approximately 20 μs. Of course this cost could be avoided by instead serialising both x and y coordinates. This would create a 64 byte ciphertext for each 31 bytes of plaintext, an expansion of 106%. We consider that to be unacceptable.

ReCrypt Operation	Time per CPU 1 block	1 KB	1 MB	1 GB	cycles/byte
Encrypt	353 μs	8.5 ms	8.9 s	2.5 hours	30.7 K
ReEnc	239 μs	7.3 ms	7.2 s	2.0 hours	26.3 K
Decrypt	328 μs	7.7 ms	7.5 s	2.2 hours	27.9 K
ReKeyGen (total)	178 μs				2.3 M

Table 4.2: Processing times for ReCrypt operations measured on a 3.8GHz CPU. 1 block represents any plaintext ≤ 31 bytes. Number of iterations: 1000 (for 1 block, 1 KB), 100 (for 1 MB) and 1 (for 1 GB). Cycles per byte given for 1MB ciphertexts.

Microbenchmarks. We built our reference implementation using the Rust [139] programming language. Our implementation is single-threaded and we measured performance on an Intel CPU (Haswell), running at 3.8GHz in turbo mode.

Table 4.2 shows wall clock times for ReCrypt operations over various plaintext sizes. As might be expected given the nature of the cryptographic operations involved, performance is far from competitive with conventional AE schemes. For comparison, AES-GCM on the same hardware platform encrypts 1 block, 1 KB, 1 MB and 1 GB of plaintext in 15 μs, 24 μs, 9 ms, and 11 s, respectively. KSS has performance determined by that of AES-GCM, while the performance of the ReCrypt scheme is largely determined by the scalar multiplications required to evaluate the PRF. Across all block sizes there is a 1000x performance cost to achieve our strongest notion of security.

Discussion. Given the performance difference, ReCrypt is best suited to very small or very valuable plaintexts (ideally, both). If the plaintext corpus is moderately or very large, cost and performance may prohibit practitioners from using ReCrypt over more performant schemes like KSS that give strictly weaker security. To make this discussion concrete, we consider two examples in more detail.

For privacy and security reasons, regulation mandates that credit card numbers must be stored in encrypted form [157]. These include the recommendation that a mechanism must be in place to rotate keys on a regular basis and in the face of known or suspected compromise. Given the sensitive nature of payment information, the naive solution of decrypting and encrypting the data to rotate keys exposes it to some risk, since it makes the data available in plaintext form for at least a period of time. Similarly, NIST guidelines [22] recommend balancing the risk introduced by the re-encryption process with the benefits offered by key rotation. On the other hand, our updatable AE schemes enable secure rotation of keys using an untrusted storage service.

Consider a payment system with, say, 1 billion credit card entries. The storage required for encrypted credit card numbers using ReCrypt is about 30 GB (assuming one block per entry). Projecting from performance measurements in Table 4.2, a full key rotation across the this dataset requires 60 CPU-hours: a significant amount of computation, but potentially not prohibitive given infrequent rotations and many available CPUs. The time to decrypt a single entry is sub-millisecond; this is small compared to processing time for a credit card transaction.

As a second example we consider long-term storage of static data, commonly known as "deep" or "cold" storage. Such data is accessed infrequently, yet data owners may still desire (or be required to) periodically rotate the encryption keys used for protecting the data. In such cases it may be more convenient for the data owner to allow the storage provider to rotate the encryption keys using a system local to the data, as opposed to the data owner retrieving the data and performing the re-encryption.

For a rough estimation of costs, we compute the cost of performing re-encryption using ReCrypt on Amazon Web Service's Elastic Compute Cloud (AWS EC2). Using the price per CPU-hour of an AWS EC2 instance of 0.05 \$/hour (USD)[4], we compute the cost to perform updates using ReCrypt as 0.10 \$/GB. For small- to medium-sized data sets or for data sets that are particularly valuable (e.g. financial information), this cost may be justified. However, for moderately-sized to large data sets, the cost may be prohibitive and clients may favour basic security schemes like KSS.

4.10 Conclusion and Open Problems

We have given a systematic study of updatable AE, providing a hierarchy of security notions meeting different real-world security requirements and schemes that satisfy them efficiently. Along the way, we showed the limitations of currently deployed approach, as represented by AE-hybrid, improved it at low cost to obtain the KSS

[4]This rate is based on the lowest per-CPU cost on AWS as of June 2017. Namely, an m4.16xlarge instance available in the AWS US-East (Ohio) region which provides 64 CPUs and is available on-demand for 3.20 \$/hour.

scheme meeting our UP-IND and UP-INT notions, identified a flaw in the BLMR scheme, repaired it, and showed how to instantiate the repaired scheme in the ROM. Through this, we arrived at ReCrypt, a scheme that is secure in our strongest security models (UP-IND, UP-INT and UP-REENC). We implemented ReCrypt and presented basic performance benchmarks for our prototype. The scheme is slower than the hybrid approaches but offers true key rotation.

Our work puts updatable AE on a firm theoretical foundation and brings schemes with improved security closer to industrial application. While there is a rich array of different security models for practitioners to chose from, it is clear that achieving strong security (currently) comes at a substantial price. Meanwhile, weaker but still useful security notions can be achieved at almost zero cost over conventional AE. It is an important challenge to find constructions which lower the cost compared to ReCrypt without reducing security. But it seems that fundamentally new ideas are needed here, since what are essentially public key operations are intrinsic to our construction.

From a more theoretical perspective, it would also be of interest to study the exact relations between our security notions, in particular whether UP-REENC is strong enough to imply UP-IND and UP-INT. There is also the question of whether a scheme that is UP-REENC is necessarily ciphertext-dependent. Finally, we reiterate the possibility of formulating updatable AE in the nonce-based setting.

Chapter 5

Tamarin Analysis of TLS 1.3

Contents

In this chapter, we apply the symbolic analysis tool Tamarin *to the in-progress TLS 1.3 specification. We detail our approach to cover the TLS 1.3 specification with unprecedented detail, closely modelling the specification. Furthermore, we provide a thorough exploration of the stated goals of the TLS specification, and our interpretation of these goals as formal properties.*

5.1 Introduction

The Transport Layer Security (TLS) protocol is the *de facto* means for securing communications on the World Wide Web. Initially released as Secure Sockets Layer (SSL) by Netscape Communications in 1995, the protocol has been subject to a number of version upgrades over the course of its 20-year lifespan. Rebranded as TLS when it fell under the auspices of the Internet Engineering Task Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [55, 182, 147, 120, 75, 21, 20], the TLS Working Group (WG) would either respond by releasing a protocol extension (a Request For Comments (RFC) intended to provide increased functionality and/or security enhancements) or by applying the appropriate "patch" to the next version of the protocol. For a more detailed analysis of the development and standardisation of TLS see [154].

Prior to the announcement of the BEAST [99] and CRIME [100] attacks of 2011 and 2012, respectively, such a strategy was valid given the frequency with which versions were updated, and the limited number of practical attacks against the protocol.

Post-2011, however, the heightened interest in the protocol and the resulting flood of increasingly practical attacks against it [10, 9, 43, 116, 99, 100, 141, 17, 148, 38, 7, 138, 109, 45, 44] rendered this design philosophy inadequate. Coupled with pressure to increase the protocol's efficiency (owing to the release of Google's QUIC Crypto [127]), the IETF started drafting the next version of the protocol, TLS 1.3, in the Spring of 2014. Unlike the development of TLS 1.2 and below, the TLS WG adopted an "analysis-prior-to-deployment" design philosophy, welcoming contributions from the academic community before official release. There have been substantial efforts from the academic community in the areas of program verification– analysing implementations of TLS [39, 42], the development of computational models– analysing TLS within Bellare-Rogaway style frameworks [96, 124, 129, 95, 105, 121], and the use of formal methods tools such as ProVerif [53] and Tamarin [176] to analyse symbolic models of TLS [14, 113, 40]. All of these endeavours have helped to both find weaknesses in the protocol and confirm and guide the design decisions of the TLS WG.

The TLS 1.3 draft specification however, has been a rapidly moving target, with large changes being effected in a fairly regular fashion. This has often rendered much of the analysis work 'outdated' within the space of few months as large changes to the specification effectively result in a new protocol, requiring a new wave of analysis.

In this work, we follow the evolution of our tool-supported symbolic verification of TLS 1.3, stretching from the initial work by Horvat for `draft-06` described in [113] all the way to the near-final draft of TLS 1.3, adding to the large effort by the TLS community to ensure that TLS 1.3 is free of the many weaknesses affecting earlier versions, and that it is imbued with security guarantees befitting such a critical protocol. We note that most of the cryptographic mechanisms in the current TLS 1.3 draft are stable, and other than fluctuations surrounding the zero-Round-Trip-Time (0-RTT) mechanism [134], we do not expect substantial changes to come.

We primarily focus on the final version of the model, covering `draft-21`. The previous work on `draft-10`, and in particular the experience acquired from building the first version of the model, was essential in producing the final model. However, in many ways the final model obsoletes the original, since it is more sophisticated, more accurate and more relevant due to the changes to the specification.

5.1.1 Contributions

Our main contributions in this work are as follows:

1. Our first complete model covers `draft-10` of the TLS specification, modelling the possible interactions of the available handshake modes. This work was published as [85].

2. Our `draft-10` model uncovered an attack on a proposed addition to the specification, resulting in a tightening of the security of TLS 1.3.

3. The next generation of model covers the latest specification of TLS 1.3 (at the time of writing, `draft-21`) that similarly considers all the possible interactions of the available handshake modes, including PSK-based resumption and 0-RTT. Its fine-grained, modular structure greatly extends and refines the coverage of our previous symbolic models. We also note that our model is highly flexible and can easily accommodate the removal of the 0-RTT mechanism, should the need arise. This work was published as [84].

4. We prove the majority of the specified security requirements of TLS 1.3, including the secrecy of session keys, perfect forward secrecy (PFS) of session keys (where applicable), peer authentication, and key compromise impersonation resistance. We also show that after a successful handshake the client and server agree session keys and that session keys are unique across handshakes.

5. We uncover a behaviour that may lead to security problems in applications that assume that TLS 1.3 provides strong authentication guarantees.

6. We provide a novel way of exhibiting the relation between the specification and our model: we provide an annotated version of the TLS 1.3 specification that clarifies which parts are modelled and how, and which parts were abstracted. This provides an unprecedented level of modelling transparency and enables a straightforward assessment of the faithfulness and coverage of our model. We anticipate that this output will be of great benefit to the academic community analysing TLS 1.3, as well as the TLS Working Group as it provides a clear and easy-to-understand mapping between the TLS 1.3 specification and a TLS 1.3 model.

All our Tamarin input files, proofs, and the annotated TLS 1.3 specification that shows the relation between the RFC and the model, can be downloaded from [12] and [16].

5.1.2 Related Work

As mentioned, there has been a great deal of work conducted in the complementary analysis spheres pertinent to TLS 1.3. Of most interest to this work are the symbolic analyses presented in [14] and [40].

The work in [14] is an analysis of TLS 1.3 by the Cryptographic protocol Evaluation towards Long-Lived Outstanding Security (CELLOS) Consortium using the ProVerif tool. Announced on the TLS WG mailing list at the start of 2016, it showed the initial (EC)DHE handshake of draft 11 to be secure in the symbolic setting. In comparison to our work, this analysis covers only one handshake mode of a draft that is now somewhat outdated.

The ProVerif models of draft 18 presented by Bhargavan et al. in [40] include most TLS 1.3 modes, and cover rich threat models by considering downgrade attacks (both with weak cryptography and downgrade to TLS 1.2). However, unlike our work, they do not consider all modes, as they do not consider the post-handshake client authentication mode. While they cover relative strong authentication guarantees (which led to the discovery of an unknown key share attack), their analysis did not uncover the potential mismatch between client and server view that we describe in Section 5.6.2.

5.1.3 Chapter Organisation

This chapter is organised as follows. In Section 5.2 we describe the TLS 1.3 protocol and the security properties claimed in the specification. Section 5.3 describes our Tamarin model and provides a few Tamarin prover fundamentals. In Section 5.4, we describe our encoding of the security guarantees, followed by Section 5.6 where we describe our results. Section 5.7 covers the relationship between our model and the specification document, discussing how we provide a website that describes our model side-by-side with the specification, giving us unprecedented modelling transparency. We conclude in Section 5.8. with a discussion of our results and future work.

5.2 TLS 1.3

In this section we provide a brief description of the TLS 1.3 protocol as is necessary for understanding our symbolic model, and we outline the claimed security properties and guarantees of the protocol.

5.2.1 New Mechanisms

The three years of effort that has gone into crafting and fine-tuning both the security and efficiency mechanisms of TLS 1.3 is readily apparent in the large structural departures from TLS 1.2. The two protocols have broadly similar goals but exhibit many differences. For example, a full TLS 1.3 handshake requires one fewer round trip before a client can transmit protected application data, and the new 0-RTT mechanism allows less sensitive application data to be sent by the client as part of its first flight of messages.

TLS 1.3 has three key exchange modes, namely, Diffie–Hellman exchange (DHE), PSK exchange, and PSK coupled with DHE. These modes enable useful features like session resumption and the transmission of early application data. Additionally, there are a number of handshake variants that allow for group renegotiation and the sending of context-dependent, optional messages. Each of these variants has different properties and offers different security guarantees.

Furthermore, TLS 1.3 has three post-handshake mechanisms covering traffic key updates, post-handshake client authentication, and the sending of new session tickets (NSTs) for subsequent resumption via a PSK. The handshake protocol maintains a rolling transcript, on which both parties must agree. This transcript takes the form of a hash value of all of the handshake messages. Post-handshake messages, however, are not included in this transcript resulting in different security properties for the post-handshake mechanisms.

We analyse all of the TLS 1.3 key exchange modes, handshake variants, and post-handshake mechanisms simultaneously, considering all possible interactions between them. We provide a brief description of these components as well as associated message flow diagrams.

Diffie–Hellman exchange (DHE)

The default mode of TLS 1.3 allows for ephemeral DH keys to be established either over a finite field or using elliptic curves.

In an initial (EC)DHE handshake, as depicted in Figure 5.1, the client sends a `ClientHello` message containing a random nonce, i.e. a freshly generated random value, and a list of symmetric algorithms. The client also sends a set of DH key shares and the associated groups, `KeyShare`, and potentially some other extensions.

Upon receipt of a `ClientHello` message, the server selects appropriate cryptographic parameters for the connection and responds with a `ServerHello` message. This message contains a server-generated random nonce, an indication of the selected parameters and potentially some other extensions. The server also sends a `KeyShare` message, along with an `EncryptedExtensions` message and optionally a `CertificateRequest` message.

The `KeyShare` contains the server's choice of group and its ephemeral DH key share. The client and server key shares are used to compute handshake and application traffic keys.

The `EncryptedExtensions` message contains material that is not necessary for determining cryptographic parameters. For instance, the draft specification lists the server name and the maximum TLS fragment length as possible values to be sent in

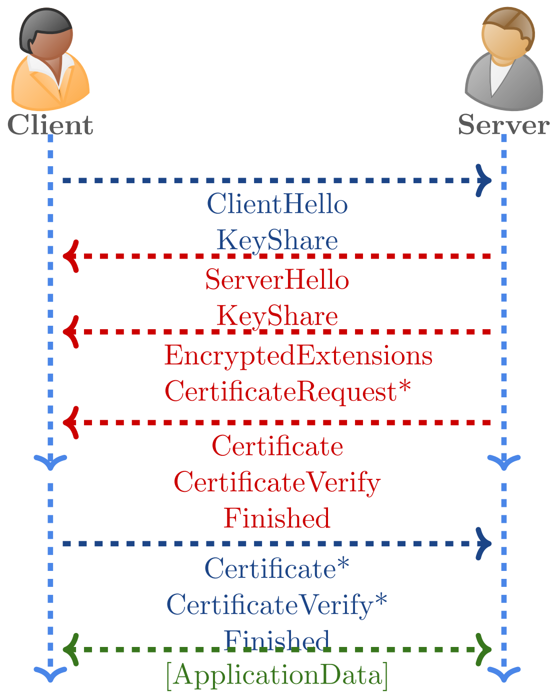

Figure 5.1: A full TLS 1.3 handshake.

this message. The `CertificateRequest` message indicates that the server requests client authentication in the mutual authentication case.

The server will also send a `Certificate` message, containing the server's certificate and a `CertificateVerify` message, which is a digital signature over the current transcript. These two messages allow the client to authenticate server. The server also sends a `Finished` message. This message is a Message Authentication Code (MAC) over the entire handshake, providing key confirmation and binding the server's identity to the computed traffic keys.

The client responds with `Certificate` and `CertificateVerify` messages, if requested, and then sends its own `Finished` message. These message flows are depicted in Figure 5.1.

Pre-shared key (PSK)

In the event that a PSK has been established, a client and a server can begin communicating without a DH exchange. This is potentially attractive for low-power environments, however, without a DHE the connection loses perfect forward secrecy (PFS). In a PSK handshake, the server authenticates via a PSK.

PSK with DHE

By combining a PSK with DHE this mode maintains PFS whilst limiting the number of expensive public key operations that the server needs to perform.

Group renegotiation

It can be the case that the groups sent by a client are not acceptable to the server. In this case, the server may respond with a `HelloRetryRequest` message. This indicates to the client which groups the server will accept, and provides the client with the opportunity to respond with an appropriate key share before returning to the main handshake.

New session ticket (NST)

After a successful handshake, the server can issue an NST at any time. These tickets create a binding to a resumption-specific secret and can be used by the client as PSKs in subsequent handshakes.

PSK binder

A PSK binder is a value that binds a PSK to the handshake where the PSK is offered by a client in a `ClientHello` message and, if the PSK was generated by a server in-band, to the handshake where it was generated. A `ClientHello` can contain multiple binders arranged in a list, where each binder is computed over a hash of the `ClientHello` message (without the binder list itself).

Session resumption and PSK

This handshake variant allows a client to use a key established out-of-band (OOB) to start a new session, or to use an NST established in a previous handshake to resume the session. This avoids the use of expensive public-key operations and in the case

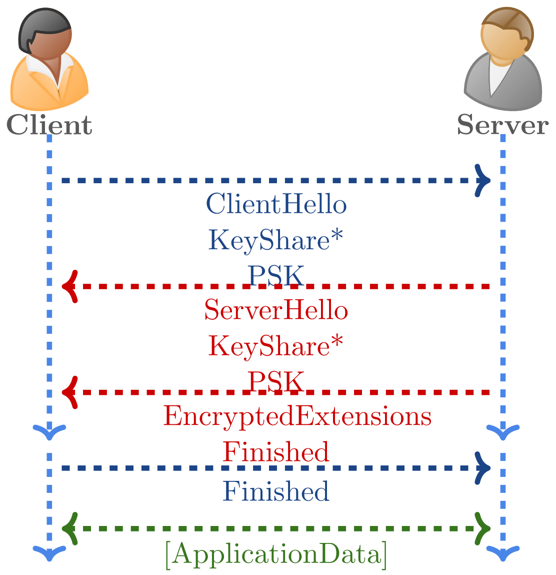

Figure 5.2: A PSK resumption handshake.

of a resumption, ties the security context of the new connection to the original connection. Note that a server may reject a resumption attempt made by a client, so the specification recommends that the client supplies an additional (EC)DHE key share with its PSK when trying to resume a session. Figure 5.2 depicts a PSK resumption handshake.

Zero round trip time (0-RTT)

A client can use a PSK to send application data in its first flight of messages, reducing the latency of the connection. As noted in the TLS 1.3 draft specification, this data is not protected against replay attacks. If the communicating entities wish to take advantage of the 0-RTT mechanism, they should provide their own replay protection at the application layer. A 0-RTT handshake is depicted in Figure 5.3.

Post-handshake client authentication

After a successful handshake, the server can send a `CertificateRequest` message. If the client responds with an acceptable certificate, then the server might authenticate the client. However, because the specification allows certificates to be rejected

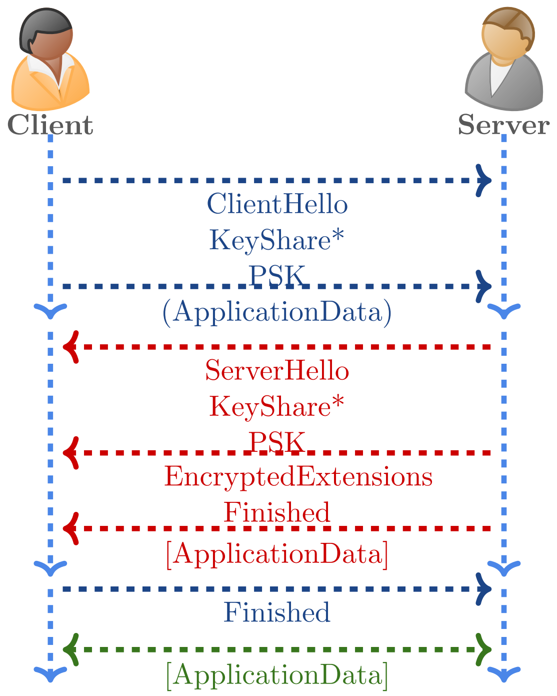

Figure 5.3: A 0-RTT handshake.

'silently', the client cannot be sure of its authentication status in general. We discuss this in greater detail in Section 5.6.2.

Key update

After a successful handshake, either party can request an application data key update. Because the read and write keys for application data are independent, either party can immediately update their write key after requesting a key update.

Key derivation

A TLS 1.3 handshake will generate a set of keys on which both the client and server agree. The specification defines a key schedule which uses the repeated application of an HMAC-based key derivation function (HKDF) [123] to combine the secret inputs with fixed labels so as to generate a set of independent keys.

The key schedule has two secret inputs, the (EC)DHE and the PSK. Depending on the handshake mode, either one or both of these will be used. The key schedule also includes the transcript hash in the key derivation. Because the transcript includes nonces, even if the secret inputs are repeated, the generated keys are guaranteed to be independent.

5.2.2 Stated Goals and Security Properties

The TLS 1.3 handshake protocol is intended to negotiate cryptographic keys via the mechanism of authenticated key exchange (AKE). These keys can then be used by the record layer to provide critical security guarantees, including confidentiality and integrity of messages. As stated in Section 5.2.1, TLS 1.3 makes use of independent keys to protect handshake messages and application data messages: protection of the handshake messages starts with the server's `EncryptedExtensions` message, and in the majority of handshake modes, protection of application data messages occurs after the transmission of the server and client `Finished` messages, respectively. In the case of a 0-RTT handshake, application data is protected with a PSK as part of the client's first flight of messages.

The TLS 1.3 specification [162, Appendix E.1] lists eight properties that the handshake protocol is required to satisfy:

1. **Establishing the same session keys**. Upon completion of the handshake, the client and the server should have established a set of session keys on which they both agree.

2. **Secrecy of the session keys**. Upon completion of the handshake, the client and server should have established a set of session keys which are known to the client and the server only.

3. **Peer authentication**. In the unilateral case, upon completion of the handshake, if a client `C` believes it is communicating with a server `S`, then it is indeed `S` who is indeed executing the server role. An analogous property for the server holds in the bilateral (mutual) authentication case.

4. **Uniqueness of session keys**. Each run of the protocol should produce distinct, independent session keys.

5. **Downgrade protection**. An active attacker should not be able to force the client and the server to employ weak cipher suites, or older versions of the TLS protocol.

6. **Perfect Forward Secrecy (PFS)**. In the case of compromise of either party's long-term key, sessions completed before the compromise should remain secure. This property is not claimed to hold in the PSK key exchange mode.

7. **Key compromise impersonation (KCI) resistance**. Should an attacker compromise the long-term key of party `A`, the attacker should not be able to use this key to impersonate an uncompromised party in communication with `A`.

8. **Protection of endpoint identities**. The identity of the server cannot be revealed by a passive attacker that observes the handshake, and the identity of the client cannot be revealed even by an active attacker that is capable of tampering with the communication.

We model six out of the eight required properties, omitting downgrade protection and the protection of endpoint identities. Also, as stated previously, 0-RTT mechanisms allow for replay of early data across sessions. We discuss the reduced 0-RTT security properties as well as the properties described above more fully in Section 5.4.

The draft specification refers to RFC 3552 [164] for an informal description of the TLS 1.3 threat model. This model assumes a Dolev-Yao attacker [93]– an attacker that can perform MITM attacks by being able to replay, insert, delete, and modify messages at will. We consider a strictly more powerful attacker, as we will explain in Section 5.4.1.

5.2.3 Changes Since `draft-10`

As mentioned, we previously modelled `draft-10` of the TLS 1.3 specification. At that point in time, there were many similarities to the latest version, with some notable differences. We provide here an overview of some of those changes to help contextualise our `draft-10` work.

First of all, the handshake modes were still in flux. In `draft-10`, the sending of early data required a client to possess a semi-static (EC)DH value of the server. This particular handshake mode was removed and replaced by a pre-shared key (PSK) 0-RTT handshake mode– early data can now only be encrypted using a PSK. In fact, the PSK mechanism has been greatly enhanced since `draft-10` with new PSK variants and binding values being incorporated in to the specification. Post-handshake authentication was officially incorporated from `draft-11` onwards and a few drafts later, post-handshake authentication was enabled to operate with the PSK handshake mode. Another change to be incorporated after `draft-10` was the inclusion of 0.5-RTT data - the server being able to send fully protected application data as part of its first flight of messages.

Since many parts of the handshake were not yet finalised, we focused on the flexibility of our model, and the ability to test out proposed changes. It is due to this approach that we were able to detect a possible attack on one of the suggested changes.

Furthermore, at the time of `draft-10`, the required security properties of TLS 1.3 were very poorly established, consisting of a loose set of recommendations as opposed to formal requirements.

5.3 Modelling the Protocol

We refer the readers to the introduction given in Section 2.3.1 for an introduction to Tamarin and its features. These features make Tamarin a good fit for the modelling and in-depth analysis of highly complex protocols such as TLS 1.3. In particular, the support for branching allowed us to model the decisions that the protocol participants can make during execution, the loops were instrumental in covering repeated connections within a single session, and the main security aspects of TLS 1.3 critically depend on Diffie-Hellman key exchange. The non-monotonic state support enabled us to model branching without having to resort to custom-tailored hacks or having to rely on the considerable over-approximation where all branches can be considered simultaneously. Lastly, the visualisations of attacks found by Tamarin provided us with a way to quickly identify potential problems, with either the protocol or our model– the graphical user interface was a great asset in guiding our TLS 1.3 verification workflow.

We defer the details of our Tamarin model to Section 5.4, and note differences to other TLS 1.3 models in the next section.

5.3.1 A Comprehensive Model

Using Tamarin's modelling framework we devised a comprehensive symbolic model of TLS 1.3 that captures the specified protocol behaviours, as well as unexpected behaviours that arise from a complex interaction of an unbounded number of sessions. Our model captures these behaviours in the presence of a powerful adversary. We defer discussion of our adversary capabilities to Section 5.4.1.

Other TLS 1.3 analyses consider the constituent parts of TLS 1.3, viewing these as separate protocols, and proceed to tie the individual proofs together with a compositionality result. For instance, [40] considers the resumption mechanism as a separate protocol in which both the client and the server take as input a symmetric value—the PSK. If the PSK remains unknown to the attacker in every execution of the resumption protocol, a gap remains to be filled before concluding that the full handshake always completes without the attacker knowing the PSK. This gap is filled by a manual compositionality proof. In our work, there is no need for such manual proofs; composition is trivially satisfied by our comprehensive model, as Tamarin considers all the possible interactions in proving each property.

`draft-06`

Our first model covered the `draft-06` version of the specification, and is covered in [113]. At that point in time, the specification was at an early stage, and did not include, for example, the 0-RTT mechanisms. This model had some notable limitations. For example, bilateral and mutual authentication modes were considered, but only separately, i.e. all sessions of the same protocol role had the same authentication status.

`draft-10`

Our `draft-10` model was one of the first analyses of the TLS 1.3 specification to cover a significant portion of the protocol.

One particularly difficult component to model was the semi-static DH key shares, which were used as part of the 0-RTT mechanism.

At the time, there was also discussion of allowing the client to authenticate after a session resumption, either in the initial handshake, or using the post- handshake authentication mechanism. We extended our `draft-10` model to include this anticipated feature, which we denoted `draft-10+`.

Characteristic of these models, and those which came before, is the use of macros to define handshake messages. For example, we defined `ClientHello` to be the pair of values `nc,pc`, representing the client's nonce and "parameters", which is a placeholder value for the handshake values from which we abstract away. The rule representing the first client message, i.e., the `ClientHello` message is depicted in Figure 5.4. The use of macros to define message chunks is to aid in producing a correct model - due to the nature of symbolic pattern matching, a typo can be disastrous, and difficult to detect.

We show the resultant state machines in Figures 5.5, 5.6. Immediately, it is clear that the TLS 1.3 state machines produce some complex transitions and considering the composition of all of these modes is an intimidating task.

The benefit of our comprehensive approach was highlighted by the potential attack we found, involving multiple handshake modes, and many handshake messages. The details of the attack are covered in Section 5.5.

Current model

In the latest version, we opted to model TLS 1.3 with a significant increase in fidelity to the draft specification. Such an approach resulted in an improved ability to capture the full functionality of TLS, as well as an even broader class of realistic attacks compared to our already fruitful `draft-10` model.

Additionally, by closely matching our model to the specification and allowing for an almost line-per-line comparison, we achieve full transparency regarding which parts of the specification we abstract away from, and which assumptions our modelling process relies on. We discuss the relation between our model and the RFC in detail in Section 5.7.

```
rule C_1:
let
    // Default C1 values
    tid = ~nc

    C   = $C
    S   = $S

    // Client Hello
    nc  = ~nc
    pc  = $pc

    // Client Key Share
    ga   = 'g'^~a

    messages = <ClientHello,ClientKeyShare>

in
    [ Fr(nc)
    , Fr(~a)
    ]
  --[ C1(tid)
    , Start(tid, C, 'client')
    , Running(C, S, 'client', nc)
    , DH(C, ~a)
    ]->
    [ St_init(C,1, tid, C, nc, pc, S, ~a, messages, 'no_auth')
    , DHExp(C, ~a)
    , Out(<C,ClientHello,ClientKeyShare>)
```

Figure 5.4: Rule C_1 in our Tamarin model of TLS 1.3 `draft-10`

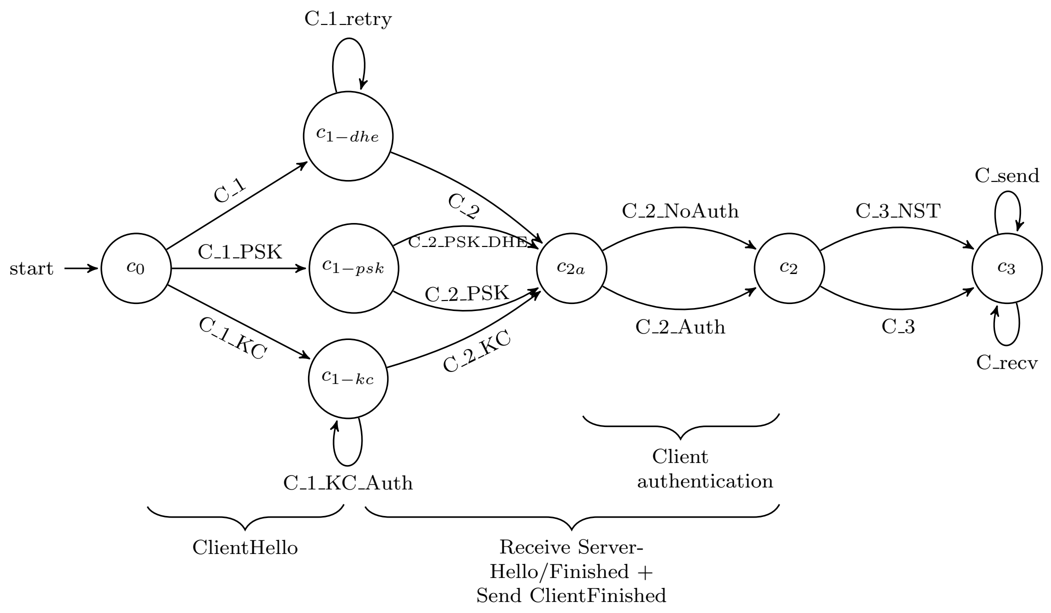

Figure 5.5: Partial client state machines for `draft-10` as modelled in our Tamarin analysis. The diagram represents the union of all the options for a client in a single execution. Not depicted are the additional transitions representing a client starting a new handshake using either a PSK established by `C_3_NST` or a `ServerConfiguration` for a 0-RTT handshake. Note that the `C_1_KC_Auth` edge may only occur once per handshake.

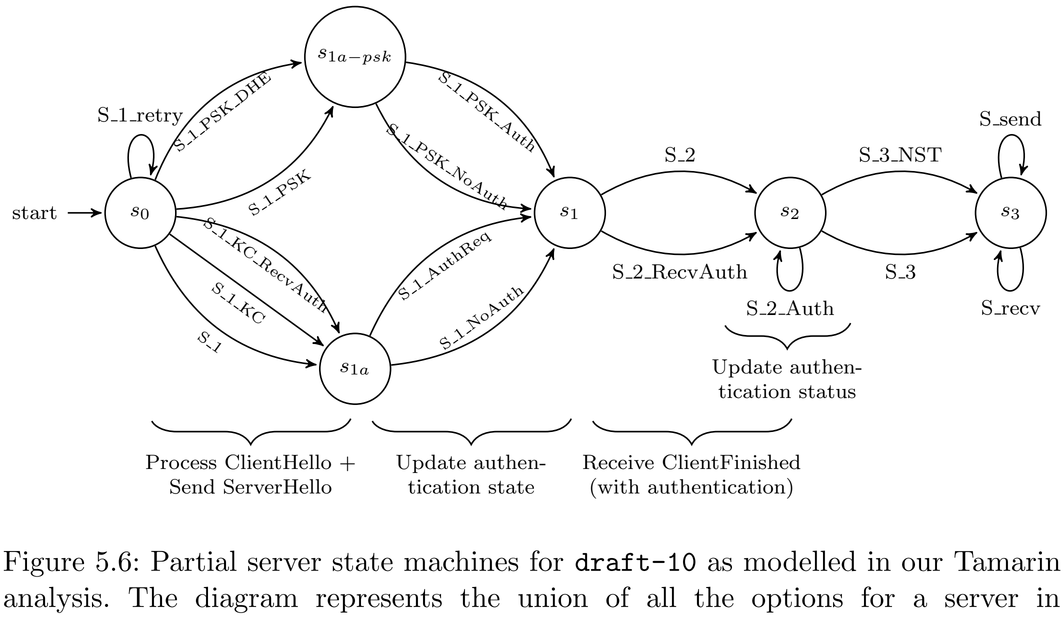

Figure 5.6: Partial server state machines for `draft-10` as modelled in our Tamarin analysis. The diagram represents the union of all the options for a server in a single execution. Not depicted are the additional transitions representing a server starting a new handshake using either a PSK established by `S_3_NST` or a `ServerConfiguration` for a 0-RTT handshake. Note that the `S_2_Auth` edge may only occur at most once per handshake.

Not only is our model more comprehensive than the TAMARIN models that precede it, it also incorporates the many changes to the TLS 1.3 specification that have materialised since the development of these models. In the following sections, we describe our modelling process, pointing out enhancements over the previous models.

5.3.2 Closely Modelling the Specification

As with our previous models, we employ the use of TAMARIN rules to model state transitions within the TLS 1.3 protocol. However, our state transitions are far more fine-grained and modular in comparison to the `draft-10` model, modelling the effective change in state as a result of transmission, receipt and processing of cryptographic parameters. For instance, a basic, initial TLS 1.3 handshake invokes up to 21 different rules and the associated state transitions before post-handshake operations can commence. These state transitions are depicted in Figure 5.7, and correspond to message flights and cryptographic processing as described in Section 5.2.1, Figure 5.1. The full state diagram can be found at the end of the chapter in Figure 5.13.

Figure 5.8 show the updated `client_hello` rule in the current model, showing a number of major changes from the previous model. For example, the messages have been significantly expanded to include extensions. Furthermore, macros are used to an even greater extent. The client state is now contained within the `ClientState` macro - a tuple of all variables.

We also note the extensive use of macros in our model, which is enabled by the `m4` preprocessor and allowed us to cover most of the specification, whilst syntactically keeping our model close to it. For example, our `ClientHello` message is a macro that expands to:

```
handshake_record('1',
    ProtocolVersion,
    ClientRandom,
    '0', // legacy_session_id
    $cipher_suites,
    '0', // legacy_compression_methods
    ClientHelloExtensions)
```

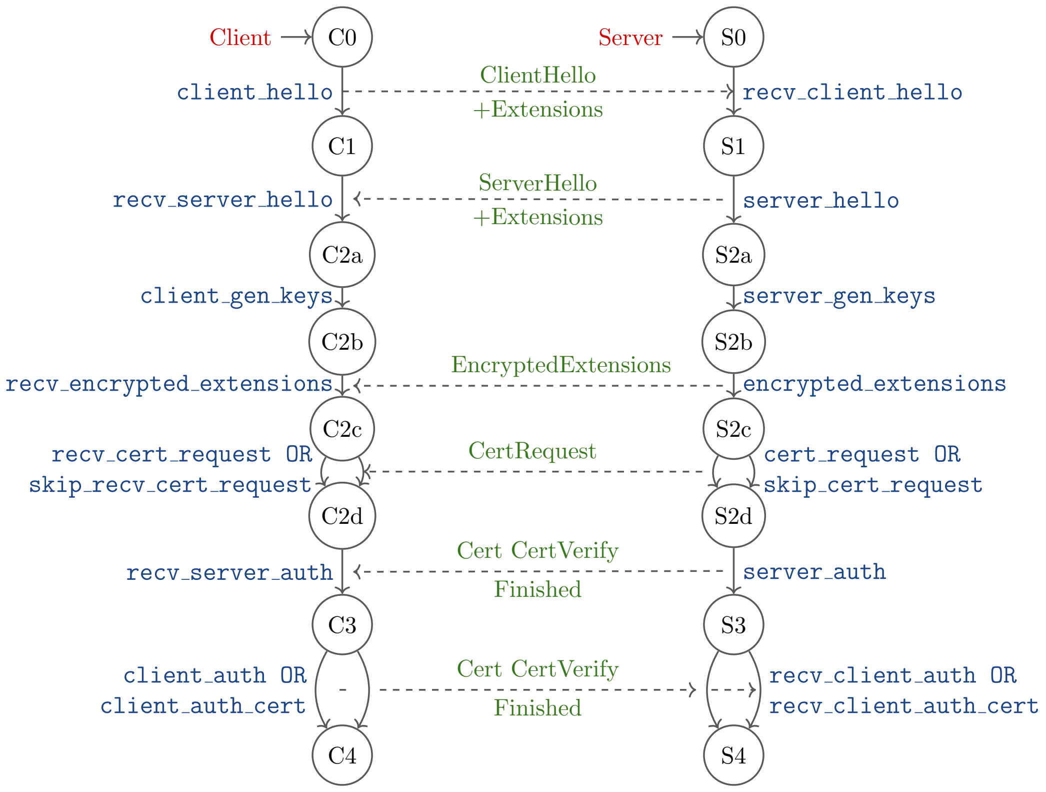

Figure 5.7: Partial state diagram for full TLS 1.3 handshake. Tamarin rules are written in snake case on edges. The messages exchanged between entities are displayed in the middle along the dashed edges. Our full model contains many more transitions, and can be found at the end of the chapter.

```
rule client_hello:
let
    // Initialise state variables to zero.
    init_state()

    // Client identity
    C = $C
    // Server identity
    S = $S
    // Client nonce
    nc = ~nc
    // We reuse the client nonce to be a thread identifier
    tid = nc

    // Group, DH exponent, key share
    g1 = $g1
    g2 = $g2
    sg = <g1, g2>
    client_sg = <g1, g2>
    g = g1
    x = ~x
    gx = g^x

    messages = <messages, ClientHello>
    es = EarlySecret
in
    [ Fr(nc), Fr(x)
    ]
  --[ C0(tid),
      Start(tid, C, 'client'),
      running_client(Identity, C),
      Neq(g1, g2),
      DH(tid, C, x)
    ]->
    [
      State(C1, tid, C, S, ClientState),
      DHExp(x, tid, C),
      Out(ClientHello)
    ]
```

Figure 5.8: Tamarin code showing the `client_hello` rule in the final TLS 1.3 model.

which reflects almost exactly how it is written in our TAMARIN files. `ClientRandom` is itself another macro, defined to be the value of the client nonce `nc`. In TAMARIN's syntax, constants are enclosed by single quotes. Constructing the model in this fashion enables a direct syntactic comparison to the specification. Previous TAMARIN models also employ macros, but the connection to the specification is much less evident. For instance, in [85] `ClientHello` is defined to be the pair of values `nc,pc`, representing the client's nonce and "parameters", which serves as a placeholder for handshake values that are abstracted away.

In our model we have tried to define cryptographic components in a way that is reminiscent of imperative programming. As in the specification, we compute the handshake secret by computing the function `HKDF-Extract(gxy,es)`, and the handshake keys are computed by applying a `Derive-Secret` function to this value. This is not strictly necessary due to the assumption of perfect cryptography, but it makes it easier to connect our model to the specification.

5.3.3 Advanced Features

In our model we capture a number of complicated interactions and logic flows inherent to the TLS 1.3 handshake, greatly improving on preceding models, adding features to the model which we consider to be 'advanced'.

Group negotiation

We model the client and the server as having a limited ability to negotiate the group used in the Diffie–Hellman key exchange.

In TAMARIN, any value can be used as a group generator. Typically, the fixed (public) constant `'g'` is used, which represents all parties agreeing to use a single group ahead of time. On receiving a key share and storing it in the variable `gx`, we simulate checking that the element resides in this group by pattern matching the value as `'g'^x = gx`. Intuitively, this corresponds to checking that $\exists x \,.\, g^x = gx$. This matches neatly with the specification, which also requires checking the DH share is not in a small subgroup.

In Tamarin's syntax, variables that are always instantiated with public values are prefixed by `$`. In our model, the client starts with a pair of public values `$g1,$g2` that represent two supported groups, and offers these to the server along with a corresponding key share for `$g1`. Similarly, the server starts with a supported group `$g`. The model allows the server to return a `HelloRetryRequest` to the client, enforcing that `$g` is not equal to `$g1`, and expects the client to return instead a key share that matches `$g2`.

This interaction enables a much greater coverage of DH key exchange with respect to previous versions of the Tamarin model, and opens up the possibility of future extensions to this work. One such extension would be to model a weak group by permitting the attacker to reveal the corresponding DH exponents.

Handshake flows

One of the most complex elements inherent to modelling TLS 1.3 is the vast number of possible state machine transitions. After a session resumption, the server can choose between using the PSK only, or using the PSK along with a DH key share. Alternatively, the server might reject the PSK entirely, and fall back to a regular handshake, or request that the client use a different group for the DH exchange. Additionally, there are several complex messages that can be sent in the post-handshake state: client authentication requests, new session tickets, and key update requests.

Since all of the above interactions can happen asynchronously, the resulting model becomes very complex and requires sophisticated handling logic. A number of complicated protocol flows, involving any number of sequential handshake modes and post-handshake extensions can, and will, transpire and we deal with this eventuality by modelling all possible handshake modes in a very modular fashion. Other models are, by and large, not capable of capturing complicated protocol flows.

5.4 Encoding the Threat Model and the Security Properties

5.4.1 Threat Model

We consider an extension of the Dolev-Yao (DY) attacker [93] as our threat model. The DY attacker has complete control of the network, and can intercept, send, replay, and delete any message. To construct a new message, the attacker can combine any information previously learnt, e.g., decrypting messages for which it knows the key, or creating its own encrypted messages. We assume perfect cryptography, which implies that the attacker cannot encrypt, decrypt or sign messages without knowledge of the appropriate keys. In order to consider different types of compromise, we additionally allow the attacker to do the following:

- compromise the long-term keys of protocol participants,
- compromise their pre-shared keys, whether created out-of-band (OOB) or through a new session ticket (NST), and
- compromise their DH values.

Note that TLS 1.3 is not intended to be secure under the full combination of all these types of compromise. For example, session key secrecy can be broken by an attacker who eavesdrops on the communication and compromises the DH values of a single protocol participant.

A natural approach is to weaken the attacker model by adding realistic constraints until either the claimed security goals of the protocol are achieved, or the corresponding attackers become weaker than the ones we expect to face in practice. This workflow requires us to express, with high granularity, exactly what needs to be protected and when each of the claimed TLS 1.3 properties can be expected to hold.

We now give our formal definitions of the TLS 1.3 security properties mentioned in Section 5.2, noting where each property is covered in our model.

5.4.2 Security Properties

We encode the claimed security properties of TLS 1.3 as lemmas in the specification language of TAMARIN. Here we discuss the relationship between the lemmas we prove in the model, and the desired properties in the specification. We note that there is some overlap between the material in the stated goals expressed in Section 5.2.2. For example, the requirement for PFS is effectively a modifier to the requirement for secret session keys. Where possible, we prove these properties via distinct lemmas to aid in the comprehension of the model. However, it is also possible to combine many of the properties into a single, more complex lemma.

Establishing the same session keys

The definition of this first property is taken from [74], where it is referred to as a consistency property. However, there is ambiguity in the circumstances that are necessary and sufficient for two protocol participants to establish the same keys. An answer to this question is typically given through the well-established practice of defining *session partnering* [34, 74, 126]. One possible way to do so is to assign session identifiers in terms of a value (or pair of values) on which the two parties agree. We opted for the least restrictive session identifier, namely the pair of nonces generated by the client and the server. Therefore, if a *partnered* client and server complete the handshake, then they must agree on session keys.

We consider this property with respect to an attacker that can compromise all session keys except for those that are identical to that of the test session, i.e. the session in which the attacker attempts to obtain information about the key [74, 126].

Secrecy of the session keys

The `secret_session_keys` lemma is used to prove property (2) in Section 5.2.2.

The `secret_session_keys` lemma we prove appears in full detail in Figure 5.9. The intuition for this lemma is that if an actor believes it has established a session key with an authenticated peer, then the attacker does not know the key. However, given the capabilities of the attacker, this will not hold without imposing some restrictions. This is why the additional clauses are required.

```
lemma secret_session_keys:
 "All tid actor peer wkey rkey peer_auth_status #i.
    SessionKey(tid, actor, peer, <peer_auth_status, 'auth'>, <wkey, rkey>)@i &
    not (Ex #r. RevLtk(peer)@r & #r < #i) &
    not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
    not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(actor, resumption_ms)@r) &
    not (Ex resumption_ms #r. RevealPSK(peer, resumption_ms)@r)
  ==> not Ex #j. K(read_key)@j"
```

Figure 5.9: The secret_session_keys lemma.

The five conditions stated in the depicted lemma are generally repeated across all lemmas, and encapsulate the basic assumption we make about our attacker. We describe them in more detail here: The first imposes the restriction that the long-term signing key of the peer is not compromised[1]. This restriction can additionally be understood to signify that the actor is communicating with an honest peer, since the attacker can effectively simulate a party when in possession of its long-term key. Furthermore, it should be noted that the attacker is still allowed to compromise the peer's long-term key (LTK) *after* the session key is established. Hence we show that the session keys achieve PFS with respect to the LTK.

The second and third clauses bar the attacker from revealing any DH exponents generated by the client or the server from before the session key was established. The attacker may reveal exponents that are generated after the session key is established.

The last two clauses specify that the attacker cannot compromise a PSK associated with either the actor or the peer. Note that the attacker is restricted from revealing these PSKs even after the session key has been established, which corresponds to the proviso in the specification that the PSK-only exchange mode does *not* provide PFS. We discuss this in more detail in Section 5.4.2.

Peer Authentication

The specification defines this property somewhat informally, as a form of authentication whereby both parties should agree on the identity of their peer. Looking at this more formally through the lens of Lowe's hierarchy of authentication [132], this definition corresponds to weak agreement. In particular, we note that this does not

[1]We remind the reader that both the client and the server are equipped with long-term signing keys, and the corresponding public key certificates, for the purposes of authentication.

```
lemma entity_authentication [use_induction, reuse]:
"All tid actor peer nonces cauth_status #i.
    CommitNonces(tid, actor, 'client', nonces)@i &
    CommitIdentity(tid, actor, 'client', peer, <cauth_status, 'auth'>)@i &
    not (Ex #r. RevLtk(peer)@r & #r < #i) &
    not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
    not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(actor, resumption_ms)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(peer, resumption_ms)@r & #r < #i)
        ==> (Ex tid2 #j. RunningNonces(tid2, peer, 'server', nonces)@j & #j < #i)"
```

Figure 5.10: The entity_authentication lemma.

imply recentness—the requirement that the peer is currently running the protocol—nor does it specify whether any other values should be agreed upon.

We initially model this property via our `entity_authentication` lemma. Entity authentication is modelled in two parts so as to capture the distinction between the bilateral (mutual) and unilateral authentication cases. Authentication in the unilateral case means that if a client completes a TLS handshake, apparently with a server, then the server previously ran a TLS handshake with the client, and they both agree on certain data values of the handshake, including the identity of the server and the nonces used. Note that this is already a stronger property than is stipulated in the specification. Here we prove non-injective agreement on the nonces, which additionally provides recentness since both parties contribute a fresh nonce to the handshake. The unilateral entity authentication lemma we prove appears in Figure 5.10.

The intuition for this lemma is that if a client believes it has agreed on a pair of nonces with a server, then the server was, at some point prior, running the protocol with those nonces. We again find the necessary restrictions on the attacker to achieve this property. The property can only hold if the attacker does not acquire any of the secrets prior to the client agreeing on nonces. While one might expect that only the legitimacy of the signing key is necessary for authentication, if the attacker is able to obtain the PSK through compromising cryptographic material, or the PSK directly, then the attacker is able to resume a session and impersonate the peer.

In addition to entity authentication, we consider a transcript agreement property, where the value agreed upon is a hash of the session transcript. This provides us with near-full agreement. However, there are a couple of notable omissions. Firstly, the protocol technically continues after the initial handshake, although none of these delayed handshake messages are included in the session transcript. Secondly, we observed that the actors do not necessarily agree on the current authentication status of the handshake, a situation we cover in more detail in Section 5.6.2.

Finally, we also prove an injective variant of mutual transcript agreement, which TLS naturally achieves by agreeing on fresh nonces. Hence, we show that TLS achieves a relatively strong authentication notion: mutual agreement on a significant portion of the state with recentness.

Uniqueness of the session keys

We prove in the straightforward way that for any two session keys generated, if they match then they must be from the same session. This holds without any restriction on the attacker, since it is a straightforward consequence of the actor generating a fresh nonce for each session. We do not prove anything about whether two session keys are related, since this trivially follows from the assumption of perfect cryptography.

Downgrade protection

The specification cites the work by Bhargavan et al. [41] for downgrade protection. This definition is not directly equivalent to any of Lowe's classical agreement methods; it only requires that both parties negotiate the *same* configuration parameters that they would do without the presence of an attacker. Specifically, we observe that agreeing on the parameters (in the sense of non-injective agreement) is sufficient to achieve this, but not necessary. Therefore, *within our model* we prove that TLS achieves downgrade protection through our authentication lemmas.

However, we note that this does not accurately capture the spirit of downgrade protection, due to the fact that we assume all cryptographic primitives are perfect and we do not model previous versions of TLS.

Forward secrecy with respect to long-term keys

The PFS property was briefly mentioned in the context of the long-term signing keys and the secrecy of session keys. However, in those cases, we did not cover the requirement for forward secrecy with regards to the PSK. We have an additional lemma `secret_session_keys_pfs` which captures that, in either a full DHE or PSK-DHE handshake, the secrecy of the session keys does not depend on the PSK remaining secret after the session is concluded.

To achieve this, we modify `secret_session_keys` as depicted in Figure 5.9, by adding a condition for the key-exchange mode, `not psk_ke_mode = psk_ke`, and loosening the restrictions on the adversary such that the `RevealPSK` action is only forbidden for any time point `#r < #i`. In proving this lemma, we show that the session keys are forward secure after a DHE.

Key Compromise Impersonation (KCI) resistance

Observant readers will notice that the only restriction on compromising long-term keys is that the peer's LTK must not be compromised. None of our security properties rely on the actor's LTK being hidden from the adversary[2]. Applying this fact to the authentication properties, therefore, additionally shows that the protocol, as given in the draft specification, achieves KCI resistance.

5.5 Enabling Client Authentication in PSK Mode

While `draft-10` did not at the time appear to permit certificate-based client authentication in PSK mode (and in particular in resumption using a PSK), we extended our model as specified in one of the proposals for this intended functionality [161].

By enabling client authentication either in the initial handshake, or with a post-handshake signature over the handshake hash, our TAMARIN analysis finds an attack. The result is a violation of client authentication, as the adversary can impersonate a client when communicating with a server.

5.5.1 The Attack

We note that the attack as described here is for the delayed authentication setting, but can easily be adapted for authentication as part of the handshake.

[2]A minor exception to this is that the adversary cannot use the actor's long-term key to impersonate the actor to themselves since in this case, the actor is also the peer.

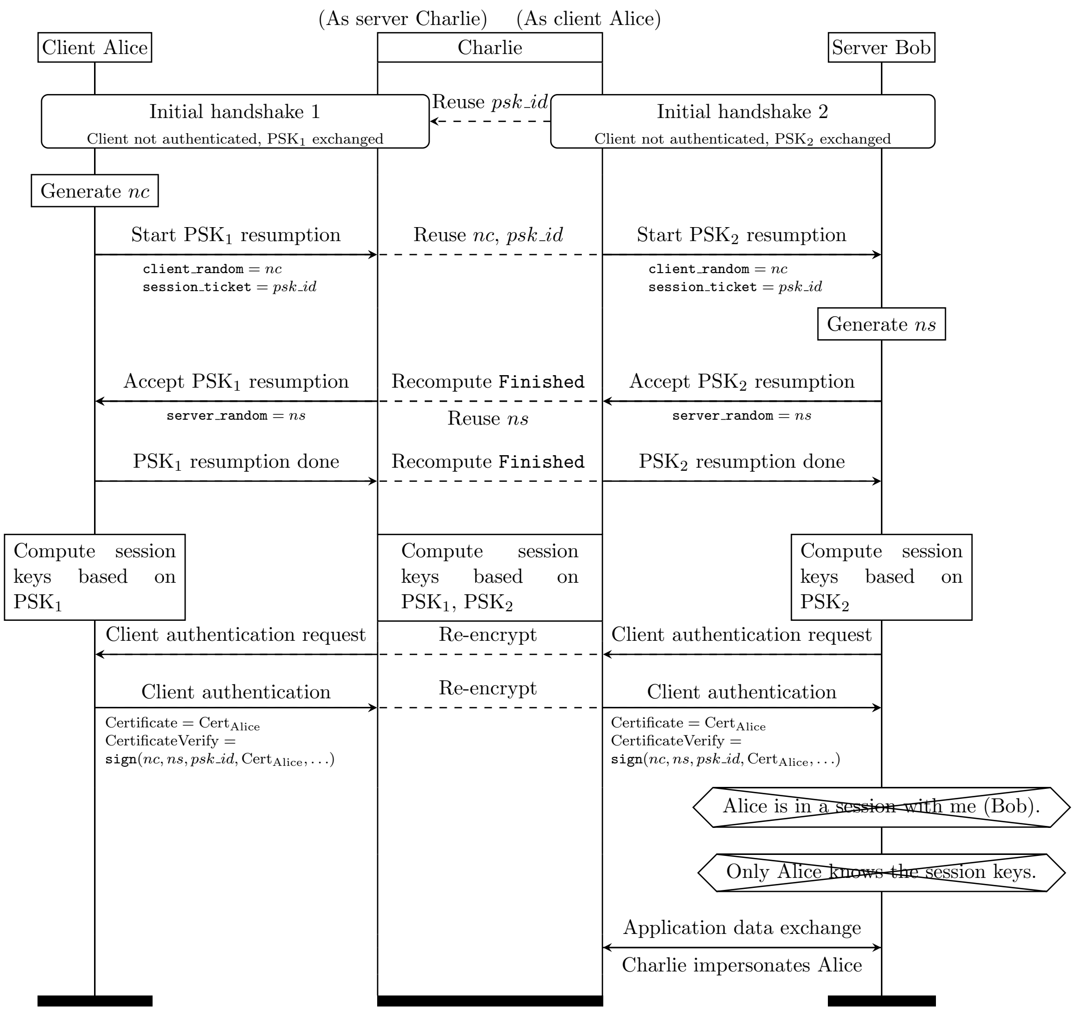
(As server Charlie)
(As client Alice)
Client Alice
Charlie
Server Bob
Initial handshake 1
Client not authenticated, PSK$_1$ exchanged
Reuse *psk_id*
Initial handshake 2
Client not authenticated, PSK$_2$ exchanged
Generate *nc*
Start PSK$_1$ resumption
client_random = *nc*
session_ticket = *psk_id*
Reuse *nc*, *psk_id*
Start PSK$_2$ resumption
client_random = *nc*
session_ticket = *psk_id*
Generate *ns*
Accept PSK$_1$ resumption
server_random = *ns*
Recompute Finished
Reuse *ns*
Accept PSK$_2$ resumption
server_random = *ns*
PSK$_1$ resumption done
Recompute Finished
PSK$_2$ resumption done
Compute session keys based on PSK$_1$
Compute session keys based on PSK$_1$, PSK$_2$
Compute session keys based on PSK$_2$
Client authentication request
Re-encrypt
Client authentication request
Client authentication
Certificate = Cert$_{Alice}$
CertificateVerify = sign(nc, ns, psk_id, Cert$_{Alice}, \ldots$)
Re-encrypt
Client authentication
Certificate = Cert$_{Alice}$
CertificateVerify = sign(nc, ns, psk_id, Cert$_{Alice}, \ldots$)
Alice is in a session with me (Bob).
Only Alice knows the session keys.
Application data exchange
Charlie impersonates Alice

Figure 5.11: Client impersonation attack on TLS 1.3 `draft-10` if delayed client authentication allowed in PSK mode. The attack involves two handshakes, two resumptions, and a client authentication request. A full explanation is given in Section 5.5.1.

We now describe the attack depicted in Figure 5.11 in more detail: Alice plays the role of the victim client, and Bob the role of the targeted server. Charlie is an active man-in-the-middle adversary, whom Alice believes to be a legitimate server. In the interest of clarity we have omitted message components and computations which are not relevant to the attack. The full attack can be reproduced using our code at [16].

The attack proceeds in three main steps, each involving different TLS subprotocols.

Step 1: Establish legitimate PSKs

In the first stage of the attack, Alice starts a connection with Charlie, and Charlie starts a connection with Bob. In both connections, a PSK is established. At this point, both handshakes are computed honestly. Alice shares a PSK denoted PSK_1 with Charlie, and Charlie shares a PSK denoted PSK_2 with Bob.

Note that Charlie ensures the session ticket (psk_id) is the same across both connections by replaying the value obtained from Bob.

Step 2: Resumption with matching freshness

In the next step, Alice wishes to resume a connection with Charlie using PSK_1. As usual, Alice generates a random nonce nc, and sends it together with the PSK identifier, psk_id.

Charlie re-uses the value nc to initiate a PSK-resumption handshake with Bob, using the same identifier, psk_id. Bob responds with a random nonce ns, and the server `Finished` message, computed using PSK_2.

Charlie now re-uses the nonce ns, and recomputes the server `Finished` message using PSK_1. Alice returns her `Finished` message to Charlie, who recomputes it using PSK_2.

At this point, Alice and Charlie share session keys (i.e., application traffic keys) derived from PSK_1, and Charlie and Bob share session keys derived from PSK_2. Note that the keys that Charlie shares with Alice and with Bob respectively, are distinct.

Step 3: Delayed client authentication

Following the resumption handshake, Charlie attempts to make a request to Bob over their established TLS channel. The request calls for client authentication, so Charlie is subsequently prompted for his certificate and verification[3]. Charlie re-encrypts this request for Alice.

To compute the verification signature, Alice uses the `session_hash` value, which is defined as the hash of all handshake messages excluding `Finished` messages. In particular, the session hash will contain nc, ns, and the session ticket psk_id.

Notice that this session hash will match the one of Charlie and Bob. Therefore, this signature will also be accepted by Bob. Hence, Charlie re-encrypts the signature for Bob, who accepts Alice's certificate and verification as valid authentication for Charlie.

Charlie has therefore successfully impersonated Alice to Bob, and even has full knowledge of the session keys. This enables Charlie to impersonate Alice in future communication with Bob, allowing him to fake messages or to access confidential resources, for instance, and violate the secrecy of messages that Bob tries to send to Alice. Thus, the attack completely breaks client authentication.

5.5.2 Underlying Cause and Mitigation

The above attack is possible due to the absence of a strong binding of the client signature to the server identity. Therefore, the attacker is able to reuse the signature it receives to impersonate the client to a server. The second component of the attack is that the attacker is able to force the two resumption sessions to have matching transcripts.

This suggests several potential ways to mitigate the attack. The most direct route would be to include the server certificate in the handshake hash. A similar fix is done in the 0-RTT case, where the server certificate is bound to the semi-static DH share. However, this is not ideal, because it complicates the out-of-band mechanism.

[3]This is one of the main use cases for the delayed client authentication mode [161].

Another potential route might be to implement an explicit authentication step as part of the PSK mechanism.

In parallel to our analysis, the TLS Working Group has proposed several modifications to `draft-10` in the move towards `draft-11`. One of these proposals is PR#316 [163] (which takes a different approach to [161]), which explicitly allows client authentication in the context that we analyse. Additionally, PR#316 redefines the client signature based on a new Handshake Context value, which includes the server `Finished` message. Intuitively, this new definition appears to address the attack because the adversary will need to force the `Finished` messages to match across the two sessions. However, the `Finished` message is bound to the PSK, which is derived from a previously authenticated session, whether using certificates or out-of band mechanisms.

Our discussions with members of the TLS Working Group reveal that they were previously not aware of the possibility of our attack, and the resulting strict necessity for a stronger binding between the client certificate and the security context that emerges from combining the PSK mode with client authentication.

5.5.3 Resulting Fix

After a few iterations, the entire PSK resumption process was entirely revamped. Now, on a resumption, the client includes a PSK binder value in the `ClientHello` message (using an extension). This indeed creates this strong binding we previously suggested; the binder value is dependent on the hash of the entire transcript, and thus the adversary will not be able to re-use the signature across two connections.

5.6 Analysis and Results

In this section we provide a detailed description of our analysis of the TLS 1.3 `draft-21`, including a discussion of our results and an exploration of an authentication anomaly uncovered by our work.

In general we find that TLS 1.3 meets the properties outlined in the specification that our modelling process was able to capture. We show that TLS 1.3 enables a client and a server to agree on secret session keys and that these session keys are unique across, as well as within, handshake instances. Our analysis shows that PFS of session keys holds in the expected situations, i.e., in the (EC)DHE and PSK+(EC)DHE handshake modes. We also show that TLS 1.3, by and large, provides the desired authentication guarantees in both the unilateral and mutual authentication cases. The situation in which this is not the case is covered in the section to follow.

We remind the reader that our model does not truly cover downgrade protection, or the protection of endpoint identities at this time. A treatment of downgrade protection across TLS protocol versions would require modelling the earlier versions of TLS in a way that is consistent with the TLS 1.3 model as developed here. To consider the downgrade protection of cipher suites, we would need to relax our current assumption of perfect cryptography through rules that, for instance, allow for an attacker to learn the payload of a particular kind of encrypted messages without knowing the key. In spite of the fact that these additional considerations would substantially complicate the model and the proof process, our model is perfectly suited to their inclusion and could form the basis of future work.

5.6.1 Positive Results

We now present our results for TLS 1.3, commenting on our proof methods and findings.

Proof strategies

For models as complex as TLS 1.3, proving lemmas in TAMARIN is a multi-stage process, and proving complex lemmas directly is often infeasible. For protocol models of this size the proof trees can become very large. TAMARIN provides a number of features that allow complex proofs to be broken down into more manageable sections. Writing sublemmas provides hints to the TAMARIN constraint solving algorithm, allowing it to solve complex sections of a larger proof directly, making the overall proof more manageable. For the TLS 1.3 model, we used several types of lemmas. Helper lemmas can be used to quickly solve repetitive sections of a larger proof without repeatedly unrolling the entire subtree. Typing lemmas provide hints to the TAMARIN engine about the potential sources of messages, reducing the branching of a proof tree. Inductive lemmas instruct Tamarin to prove the lemmas inductively, allowing us to break out of loops in the protocol, which otherwise can produce infinite proof trees. Proving the main properties of TLS 1.3 required many helper lemmas, of all of these types.

The Tamarin engine can also use heuristics to auto-prove lemmas, which proved invaluable in quickly re-proving large sections of properties after making changes to the model. By investing time in writing auto-provable sublemmas, we could flexibly incorporate changes made to the specification without having to restart our analysis from scratch.

The more complex lemmas used in our analysis of TLS 1.3, however, required manual proving in the Tamarin interactive prover. We note that by manual proving in this context we mean manually guiding the Tamarin prover through a proof by using the Tamarin graphical user interface.

Using the `m4` preprocessor to generate restricted subsets of the model we were able to prototype lemmas in a simpler environment without expending unnecessary effort. To give an indication of the number of helper lemmas required, and the relationship between all of our lemmas, we have constructed a 'lemma map', displayed in Figure 5.12. The map also indicates which lemmas were auto-proved by Tamarin, and which ones needed manual guidance for Tamarin to prove them.

In total, the most recent modelling effort represents approximately three months worth of work. However, the vast majority of that was the process of writing lemmas to break down the overall proving effort into smaller, autoprovable chunks. With these lemmas in place, proving the entire model took about one week of work, and significant computing resources. The model itself takes over 10GB RAM just to load, and can easily consume 100GB RAM in the course of a proof. In one instance, an automatically-computed proof was almost one million lines long. Once the proofs have been produced, they can be verified in the space of about a day, although still requiring a vast amount of RAM.

Findings

We summarise our results in Table 5.1. For each property discussed in Sections 5.2 and 5.4, we indicate our findings. We use $*$ to indicate that the property holds in most situations. Cases in which the property does not hold to the expected degree, are covered in sections to follow. We also list the applicable TAMARIN lemma(s).

Property proven	**Lemma(s)**
(1) Same session keys	`session_key_agreement`
(2) Secret session keys	`secret_session_keys`
(3) Peer authentication*	`entity_authentication` `mutual_entity_authentication`
(4) Unique session keys	`unique_session_keys`
(6) Perfect forward secrecy	`secret_session_keys_pfs`
(7) Key compromise impersonation	`entity_authentication` `mutual_entity_authentication`

Table 5.1: TLS 1.3 TAMARIN results

5.6.2 Possible Mismatch Between Client and Server View

During the development of our model, and in particular the analysis of the post-handshake client authentication, we encountered a possible behaviour that suggested that TLS 1.3 fails to meet certain strong authentication guarantees.

While there are many definitions of authentication, the common thread among strong authentication guarantees is that both parties share a common view of the session, i.e. that they agree on exchanged data, keys, etc. During our analysis of the post-handshake client authentication, it became apparent that the client does not receive any explicit confirmation that the server has successfully received the client's response. Due to the asynchronous nature of the post-handshake client authentication, the client may keep receiving data from the server, and will not be able to determine if the server has received its authentication message. As a consequence, the client cannot be sure whether the server sent the data under the assumption that the client is authenticated.

We formally modelled this property by adding a variable to the client and the server that records the current status of the connection, and in particular, if the connection is unilaterally or mutually authenticated. We discovered that even when the server

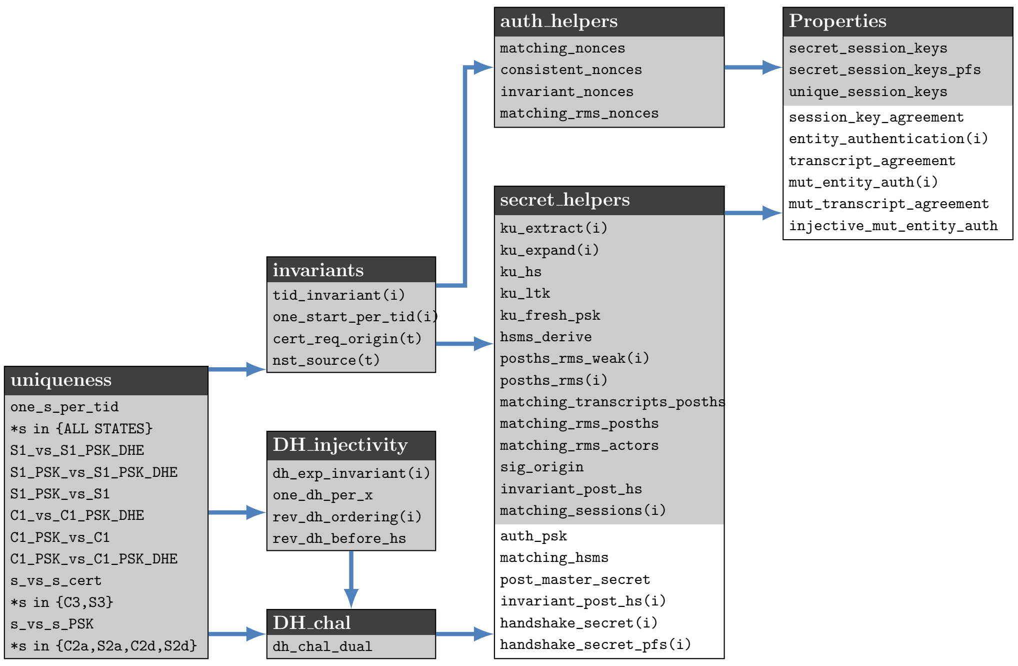

Figure 5.12: Lemma Map. Lemma names with a white background indicate where manual interaction via the TAMARIN visual interface was required. The remaining lemmas were automatically proven by TAMARIN, without manual interaction. An arrow from one category to another implies that the proof of the latter depends on the former. The **Properties** box contains the main TLS 1.3 properties.

asks for a post-handshake client authentication, and the client responds, the client cannot be sure that the server considers the channel to be mutually authenticated.

In concurrent work by Bhargavan et al. [40], a similar issue was uncovered for the 0.5-round trip time (RTT) case. A discussion with the TLS 1.3 working group revealed that an equivalent problem also exists within the main handshake. During the main handshake, the server can request a client certificate, and may decide to reject the certificate (for example because it violates certain domain-specific policies), *but still continue with the connection* as if the certificate were accepted. Therefore, the client cannot be sure (after what appears to be a main handshake with mutual authentication) that the server considers the client to be authenticated. Thus, this phenomenon leaves the client in the dark about whether or not the server considers it to be authenticated, even though the server asked for a certificate and the client supplied it.

To see why this may become a problem at the application level, consider the following application. Imagine a client and a server that implement TLS 1.3, where the server has the following policy: any data received over a mutually authenticated connection are stored in a secure database; all data received over connections where the client is not authenticated are stored in an insecure log. The client connects, the server requests a certificate, which the client duly provides, but the server rejects and continues regardless. Since the server rejected the certificate, it continues to store incoming messages in the insecure log. However, the client may assume it has been authenticated, and start sending sensitive data, which ends up in the insecure log.

The TLS working group has decided not to fix this behaviour for TLS 1.3, and has not introduced any mechanism that informs the client of the server's view of the client's authentication status. If a client wants to be sure that the server considers it to be authenticated, this needs to be dealt with at the application layer. We anticipate that some client applications will incorrectly assume that sending a client certificate and obtaining further server messages indeed guarantees that the server considers the connection to be mutually authenticated. As we have shown, this is not the case in general, and may lead to serious security issues despite there being no direct violation of the specified TLS 1.3 security requirements.

5.7 The Relation Between Our Model and the TLS 1.3 Specification

While there have been many academic analyses of various drafts of TLS 1.3 [40, 96, 14, 124, 129, 95, 105, 121, 113], they all (explicitly or implicitly) consider only part of the specification. Most analyses, even those that claim to be "complete" do not consider all possible modes, and many manual cryptographic analyses consider modes only in isolation (and not their interaction). This is caused by the inherent complexity of analysing TLS 1.3 and is not a problem in itself; rather, it justifies the need for multiple approaches.

However, we are of the opinion that readers, regardless of whether or not they are experts in the field, should be able to easily deduce the exact coverage of a given analysis. To ensure this, we provide an unprecedented level of transparency concerning the relationship between our model and the RFC (the draft specification) by creating a website [12] that contains an annotated version of the RFC.

For example, the website shows how the concrete data structures of TLS 1.3 are mapped into abstract term structures. Additionally, we annotate the prose, describing the possible behaviours so as to indicate which Tamarin rules model them. The annotations also show exactly which details we do not model (and often list the reasons why).

We used these annotations ourselves during the development of our model to keep track of the parts of the specification that we had already modelled, and how we modelled them, which also simplified the task of keeping track of updates to the specification, something which proved incredibly useful given the rapid pace at which the draft specification would undergo changes.

Our annotated RFC has a number of desirable features:

- Readers can check which parts we abstracted, and how, without having to reinvent the mapping between the Tamarin model and the RFC themselves. In other words, one can read through our website to see what is covered, and how it is covered, without having to understand Tamarin's formalism.

- If the specification is updated or changed, we can immediately track where the model should be changed.

We encourage other analyses of TLS 1.3 to follow a similar transparent approach, which would help the community to better understand which details from the specification might still need to be covered. We envision this will enable a faster convergence of confidence in all the details of the standard.

5.8 Conclusions

In this work we modelled the current draft of the TLS 1.3 specification within the symbolic analysis framework of the Tamarin prover, and used the tool to verify the majority of the security guarantees that TLS 1.3 claims to offer its users.

We focus on ruling out complex interaction attacks by considering an unbounded number of concurrent connections, and all of the TLS 1.3 handshake modes. We cover both unilateral and mutual authentication, as well as session key secrecy in all of the TLS 1.3 handshake modes with respect to a Dolev-Yao attacker. We also capture more advanced security properties such as perfect forward secrecy and key compromise impersonation. Our Tamarin model covers substantially more interactions than previous analyses due to its modularity.

Besides verifying that draft 21 of the TLS 1.3 specification meets the claimed security properties in most of the handshake modes and variants, we also discover an unexpected authentication behaviour which may have serious security implications for implementations of TLS 1.3. This unexpected behaviour, at a high level, implies that TLS 1.3 provides no direct means for a client to determine its authentication status from the perspective of a given server. As a server may treat authenticated data differently to unauthenticated data, the client may end up in position in which its sensitive data gets processed as non-sensitive data by the server.

During the course of our analysis we also developed a line-by-line modelling aide that accurately captured which parts of the specification we were able to model, and which parts were abstracted. This artefact allows us to easily assess the faithfulness and coverage of our model, and also makes our model highly amenable to all kinds of extensions, especially with respect to the security properties and threat model.

We expect that this artefact may serve as a comprehensive informational aide to academic researchers and well as the TLS Working Group.

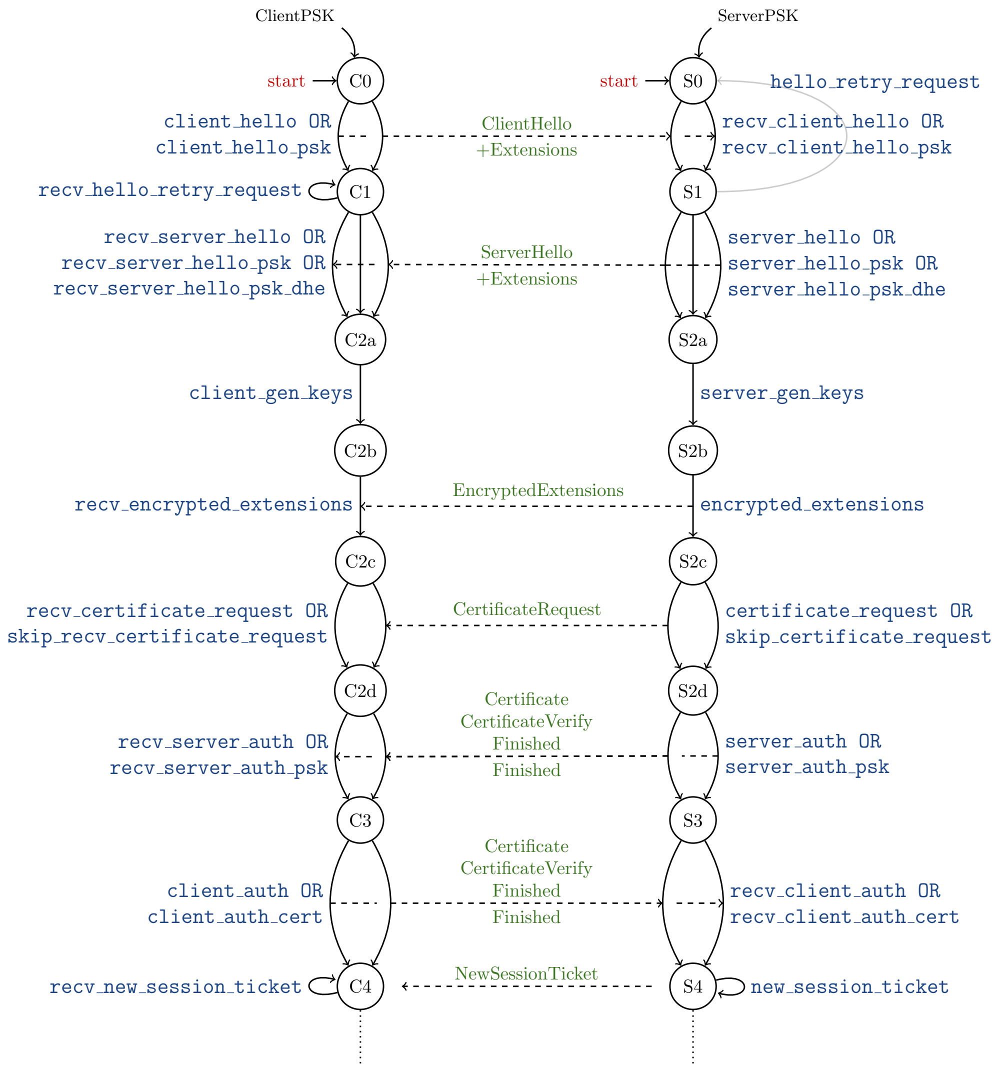

Figure 5.13: Part 1 of the full state diagram for Tamarin model, showing all rules covered in the initial handshake (excluding rules dealing with record layer).

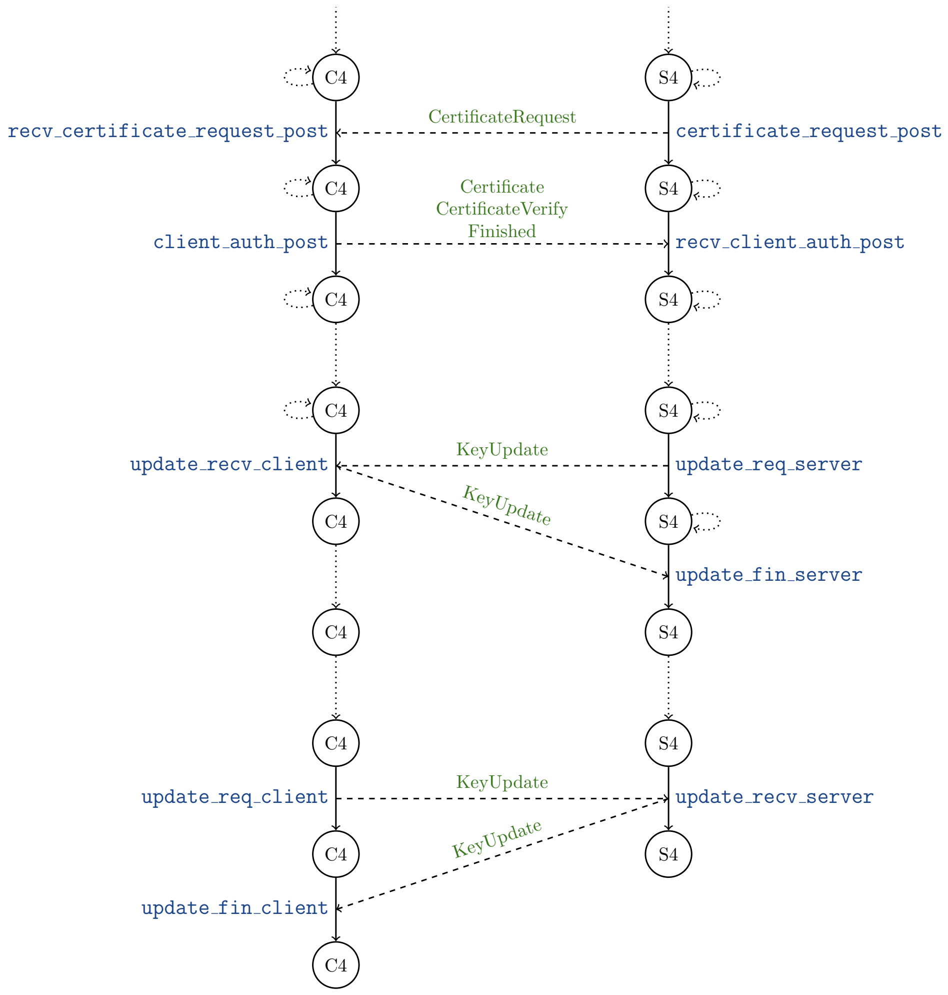

Figure 5.14: Part 2 of the full state diagram for TAMARIN model, showing all post-handshake rules covered.

Chapter 6

Concluding Remarks

In this thesis, we have covered a range of real-world cryptographic scenarios and protocols, from password storage, to the ubiquitous TLS. A common theme throughout has been understanding how to apply cryptographic analysis techniques in order to *prove* that a certain design meets its stated goals.

Unravelling this short statement reveals a number of complex interactions. For example, there is a constant back-and-forth between adequately stating the goals of an application, and designing a specific implementation to meet those goals.

In the case of key rotation for authenticated encryption in Chapter 4, there was already a pre-existing belief that key rotation was an essential part of secure data storage, but without any formal understanding of what constituted a secure re-encryption. By formally describing security goals for this scenario, we were able to evaluate existing schemes, understand their flaws and propose new schemes. This process helped to elucidate certain requirements that sounded obvious, but were difficult to precisely state, such as "a re-encryption should look like a fresh encryption".

In the course of designing our AE models, we uncovered an interesting question: whether the scheme should rely on the security of ciphertext headers. For the sake of designing a strong model, we opted to permit the adversary to recover ciphertext headers, modelling a temporary compromise of the "client" service. Taking the time to consider all possible adversarial behaviours results in asking these kinds of questions, for example, under what model does it make sense for an adversary to obtain some state or variable?

Similarly, the design of the TLS specification has included lengthy discussions on what the requirements and goals of TLS are, and to what extent it meets those objectives. The development of TLS has led to new research avenues, aiming to understand how each small, and seemingly innocuous, component can impact the security of the protocol. As the protocol evolves to meet these new requirements, so does the research community in producing new, stronger standards for it to attain.

On the other hand, this thesis has also scratched the surface of what the available analysis mechanisms are, and how they can contribute in different ways to the analysis of a protocol or specification. In Chapters 3 and 4, we used traditional computational techniques to prove our implementations were secure. However, in Chapter 5, we instead relied on formal methods in the form of the Tamarin prover to analyse the latest TLS 1.3 draft.

Comparing the two approaches as they were used in this thesis, the computational analysis provided us a very "high-level" view of the security of a protocol. We consider the base elements, and prove their security is based on well-established (or even brand new) cryptographic assumptions. Both Pythia and ReCrypt relied on the security of key-homomorphic PRFs, and ultimately reduced to variants of the DDH assumption.

Constructing a concrete implementation of ReCrypt in Chapter 4 showed how sometimes this high-level analysis leads to complications in practice. When building a protocol out of primitives such as key-homomorphic PRFs, it is common to make assumptions such as "messages are members of the group $\mathcal{X}$". In practice, we often work with bitstrings, and must prove the security of the round trip from a bitstring to a group element, applying the protocol, and back to a bitstring. Hence we also needed to propose an encoding function from bitstrings to group elements. It was not obvious here what was required from the encoding function, and required additional thought and investigation.

This exemplifies how, despite having a computational proof, there are still many ways in which a protocol can be theoretically unsound. To leave the description of ReCrypt without detailing this necessity would almost certainly lead to a inexperienced developer implementing an unsound protocol.

In contrast, the symbolic approach using Tamarin cannot necessarily provide guarantees on the same level as the computational proofs. Indifferentiability for example is currently being added to Tamarin, but seems unobtainable for complex protocols in the short-term. Where symbolic analysis excels, is the holistic analysis of the protocol. In Chapter 3, we detailed an entire API for use in the password storage service. However, the computational analysis only covered one small component of the service - the PRF queries. This leaves an analytical chasm, between a production-ready implementation and the original prototype, and symbolic analysis tools can help to fill this gap.

Furthermore, the symbolic approach helps to identify any incorrect assumptions made about the model or scenario. A priori, nothing is held to be true, and thus Tamarin is able to invoke surprising combinations of behaviours which may not have been possible before. In [85], we found an attack on an early version of TLS not through directing Tamarin to find the attack, but because while trying to prove basic properties about the handshake, it became clear that there was an equation which could not be satisfied. This was part of a complex inductive proof, crossing multiple handshake modes and involving many messages. It is possible that a computational analysis may have missed such an attack, by making a simplifying assumption across handshake modes which would ignore this possibility.

Bibliography

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. “The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES”. In: *Topics in Cryptology – CT-RSA 2001*. Ed. by David Naccache. Vol. 2020. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2001, pp. 143–158.

[2] Masayuki Abe and Tatsuaki Okamoto. “Provably Secure Partially Blind Signatures”. In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 271–286.

[3] Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. “Cryptographic Agility and Its Relation to Circular Encryption”. In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. French Riviera: Springer, Heidelberg, Germany, May 2010, pp. 403–422.

[4] Ting Yu, George Danezis, and Virgil D. Gligor, eds. *ACM CCS 12: 19th Conference on Computer and Communications Security*. Raleigh, NC, USA: ACM Press, Oct. 2012.

[5] Indrajit Ray, Ninghui Li, and Christopher Kruegel: eds. *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Denver, CO, USA: ACM Press, Oct. 2015.

[6] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel: Denver, CO, USA: ACM Press, Oct. 2015, pp. 5–17.

[7] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*. Denver, Colorado, USA, 2015.

[8] Martin R Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of learning with errors". In: *Journal of Mathematical Cryptology* 9.3 (2015), pp. 169–203.

[9] Nadhem J. AlFardan and Kenneth G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. 2013, pp. 526–540.

[10] Nadhem J. AlFardan and Kenneth G. Paterson. "Plaintext-Recovery Attacks Against Datagram TLS". In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. 2012.

[11] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. "How to Securely Release Unverified Plaintext in Authenticated Encryption". In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 105–125.

[12] Anonymous. *Source files and annotated RFC for TLS 1.3 rev 20 analysis*. dropbox.com/sh/aiaqgbnt37m5yza/AABmQ__X1lBlxALEJactJAS6a. 2017.

[13] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. *Fine grain Cross-VM Attacks on Xen and VMware are possible!* Cryptology ePrint Archive, Report 2014/248. http://eprint.iacr.org/2014/248. 2014.

[14] Kenichi Arai and Shin'ichiro Matsuo. *Formal Verification of TLS 1.3 Full Handshake Protocol Using ProVerif*. TLS mailing list post. Internet Engineering Task Force, Feb. 2016.

[15] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient LIbrary for Cryptography*. https://github.com/relic-toolkit/relic.

[16] *Archive with TLS 1.3 Rev 10 models and property specifications for the Tamarin prover*. http://tls13tamarin.github.io/TLS13Tamarin/. 2015.

[17] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. “DROWN: Breaking TLS with SSLv2”. In: *25th USENIX Security Symposium*. Aug. 2016.

[18] AWS. *Protecting Data Using Client-Side Encryption*. http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingClientSideEncryption.html.

[19] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. “Password-protected secret sharing”. In: *ACM CCS 11: 18th Conference on Computer and Communications Security*. Ed. by Yan Chen, George Danezis, and Vitaly Shmatikov. Chicago, Illinois, USA: ACM Press, Oct. 2011, pp. 433–444.

[20] Gregory V. Bard. “A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL”. In: *SECRYPT 2006, Proceedings of the International Conference on Security and Cryptography, Setúbal, Portugal, August 7-10, 2006, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*. 2006, pp. 99–109.

[21] Gregory V. Bard. “The Vulnerability of SSL to Chosen Plaintext Attack”. In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 111.

[22] Elaine Barker. “Recommendation for key management Part 1: General”. In: *NIST special publication 800-57 Part 1 Revision 4*. 2016.

[23] Paulo S. L. M. Barreto and Michael Naehrig. “Pairing-Friendly Elliptic Curves of Prime Order”. In: *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. Lecture Notes in Computer Science. Kingston, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2006, pp. 319–331.

[24] Guy Barwell, Daniel Page, and Martijn Stam. “Rogue Decryption Failures: Reconciling AE Robustness Notions”. In: *15th IMA International Conference on Cryptography and Coding*. Ed. by Jens Groth. Vol. 9496. Lecture Notes in Computer Science. Oxford, UK: Springer, Heidelberg, Germany, Dec. 2015, pp. 94–111.

[25] David A. Basin, Jannik Dreier, and Ralf Sasse. “Automated Symbolic Proofs of Observational Equivalence”. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel: Denver, CO, USA: ACM Press, Oct. 2015, pp. 1144–1155.

[26] Mihir Bellare. *Introduction to Modern Cryptography (lecture notes).*

[27] Mihir Bellare. “Practice-Oriented Provable-Security (Invited Lecture)”. In: *ISW’97: 1st International Workshop on Information Security.* Ed. by Eiji Okamoto, George I. Davida, and Masahiro Mambo. Vol. 1396. Lecture Notes in Computer Science. Tatsunokuchi, Japan: Springer, Heidelberg, Germany, Sept. 1998, pp. 221–231.

[28] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. “A Concrete Security Treatment of Symmetric Encryption”. In: *38th Annual Symposium on Foundations of Computer Science.* Miami Beach, Florida: IEEE Computer Society Press, Oct. 1997, pp. 394–403.

[29] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. “DupLESS: server-aided encryption for deduplicated storage”. In: *USENIX Security.* USENIX. 2013.

[30] Mihir Bellare and Tadayoshi Kohno. “A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications”. In: *Advances in Cryptology – EUROCRYPT 2003.* Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 491–506.

[31] Mihir Bellare and Chanathip Namprempre. “Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm”. In: *Advances in Cryptology – ASIACRYPT 2000.* Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, Dec. 2000, pp. 531–545.

[32] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. “The One-More-RSA-Inversion Problems and the Security of Chaum’s Blind Signature Scheme”. In: *Journal of Cryptology* 16.3 (June 2003), pp. 185–215.

[33] Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. “Multi-instance Security and Its Application to Password-Based Cryptography”. In: *Advances in Cryptology – CRYPTO 2012.* Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 312–329.

[34] Mihir Bellare and Phillip Rogaway. “Entity authentication and key distribution”. In: *Annual International Cryptology Conference.* Springer. 1993, pp. 232–249.

[35] Mihir Bellare and Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by V. Ashby. Fairfax, Virginia, USA: ACM Press, Nov. 1993, pp. 62–73.

[36] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 2006, pp. 207–228.

[37] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. “Elligator: elliptic-curve points indistinguishable from uniform random strings”. In: *ACM CCS 13: 20th Conference on Computer and Communications Security*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. Berlin, Germany: ACM Press, Nov. 2013, pp. 967–980.

[38] 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: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 2015, pp. 535–552.

[39] K. Bhargavan, N. Kobeissi, and B. Blanchet. “ProScript TLS: Building a TLS 1.3 Implementation with a Verifiable Protocol Model”. Presentation. Presented at TRON 1.0, San Diego, CA, USA, February 21. 2016.

[40] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. *Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate*. Tech. rep. INRIA, 2017.

[41] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. “Downgrade resilience in key-exchange protocols”. In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 506–525.

[42] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Karim Zinzindohoué. *Implementing and Proving the TLS 1.3 Record Layer*. Tech. rep. http://eprint.iacr.org/2016/1178. INRIA, 2016.

[43] 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: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 2014, pp. 98–113.

[44] Karthikeyan Bhargavan and Gaëtan Leurent. “On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: ACM, 2016, pp. 456–467. ISBN: 978-1-4503-4139-4.

[45] Karthikeyan Bhargavan and Gaëtan Leurent. “Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH”. In: *Network and Distributed System Security Symposium–NDSS 2016*. 2016.

[46] *Technical background of version 1 Bitcoin addresses*. https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses.

[47] *Bitcoin Wiki, “Weaknesses”*. https://en.bitcoin.it/wiki/Weaknesses.

[48] John Black, Phillip Rogaway, and Thomas Shrimpton. “Encryption-Scheme Security in the Presence of Key-Dependent Messages”. In: *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Kaisa Nyberg and Howard M. Heys. Vol. 2595. Lecture Notes in Computer Science. St. John’s, Newfoundland, Canada: Springer, Heidelberg, Germany, Aug. 2003, pp. 62–75.

[49] Simon R. Blackburn and Sam Scott. “The discrete logarithm problem for exponents of bounded height”. In: *LMS Journal of Computation and Mathematics* 17 (Special Issue A Jan. 2014), pp. 148–156. ISSN: 1461-1570.

[50] Bruno Blanchet. “An Efficient Cryptographic Protocol Verifier Based on Prolog Rules”. In: *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*. CSFW ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 82–.

[51] Bruno Blanchet. “Security protocol verification: Symbolic and computational models”. In: *Proceedings of the First international conference on Principles of Security and Trust*. Springer-Verlag. 2012, pp. 3–29.

[52] Bruno Blanchet and David Cadé. *CryptoVerif: Cryptographic protocol verifier in the computational model*. 2011.

[53] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *Proverif 1.96: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial List of Figures*. 2016.

[54] Matt Blaze, Gerrit Bleumer, and Martin Strauss. "Divertible Protocols and Atomic Proxy Cryptography". In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 127–144.

[55] Daniel Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1". In: *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*. 1998, pp. 1–12.

[56] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003, pp. 31–46.

[57] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. "On Symmetric Encryption with Distinguishable Decryption Failures". In: *Fast Software Encryption – FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Mar. 2014, pp. 367–390.

[58] Dan Boneh. "The decision Diffie-Hellman problem". In: *Third Algorithmic Number Theory Symposium (ANTS)*. Vol. 1423. Lecture Notes in Computer Science. Invited paper. Springer, Heidelberg, Germany, 1998.

[59] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 213–229.

[60] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. "Key Homomorphic PRFs and Their Applications". In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 410–428.

[61] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. *Key Homomorphic PRFs and Their Applications.* Cryptology ePrint Archive, Report 2015/220. http://eprint.iacr.org/2015/220. 2015.

[62] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short Signatures from the Weil Pairing”. In: *Advances in Cryptology – ASIACRYPT 2001.* Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 514–532.

[63] Dan Boneh and Brent Waters. “Constrained Pseudorandom Functions and Their Applications”. In: *Advances in Cryptology – ASIACRYPT 2013, Part II.* Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. Lecture Notes in Computer Science. Bengalore, India: Springer, Heidelberg, Germany, Dec. 2013, pp. 280–300.

[64] *Brainwallet.* https://en.bitcoin.it/wiki/Brainwallet.

[65] Emmanuel Bresson, Jean Monnerat, and Damien Vergnaud. “Separation Results on the “One-More” Computational Problems”. In: *Topics in Cryptology – CT-RSA 2008.* Ed. by Tal Malkin. Vol. 4964. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2008, pp. 71–87.

[66] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. “Efficient Indifferentiable Hashing into Ordinary Elliptic Curves”. In: *Advances in Cryptology – CRYPTO 2010.* Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 237–254.

[67] Daniel R. L. Brown. *Irreducibility to the One-More Evaluation Problems: More May Be Less.* Cryptology ePrint Archive, Report 2007/435. http://eprint.iacr.org/2007/435. 2007.

[68] Jan Camenisch and Anna Lysyanskaya. “An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation”. In: *Advances in Cryptology – EUROCRYPT 2001.* Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Innsbruck, Austria: Springer, Heidelberg, Germany, May 2001, pp. 93–118.

[69] Jan Camenisch, Anna Lysyanskaya, and Gregory Neven. “Practical yet universally composable two-server password-authenticated secret sharing”. In: *ACM CCS 12: 19th Conference on Computer and Communications Security.* Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. Raleigh, NC, USA: ACM Press, Oct. 2012, pp. 525–536.

[70] Jan Camenisch, Gregory Neven, and abhi shelat. "Simulatable Adaptive Oblivious Transfer". In: *Advances in Cryptology – EUROCRYPT 2007*. Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Heidelberg, Germany, May 2007, pp. 573–590.

[71] Jan Camenisch and Markus Stadler. *Proof Systems for General Statements about Discrete Logarithms.* Technical Report No. 260, Dept. of Computer Science, ETH Zurich. 1997.

[72] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited (Preliminary Version)". In: *30th Annual ACM Symposium on Theory of Computing.* Dallas, TX, USA: ACM Press, May 1998, pp. 209–218.

[73] Ran Canetti and Susan Hohenberger. "Chosen-ciphertext secure proxy re-encryption". In: *ACM CCS 07: 14th Conference on Computer and Communications Security.* Ed. by Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson. Alexandria, Virginia, USA: ACM Press, Oct. 2007, pp. 185–194.

[74] Ran Canetti and Hugo Krawczyk. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels". In: *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding.* 2001, pp. 453–474.

[75] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. "Password Interception in a SSL/TLS Channel". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings.* 2003, pp. 583–599.

[76] David Cash, Matthew Green, and Susan Hohenberger. "New Definitions and Separations for Circular Security". In: *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography.* Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2012, pp. 540–557.

[77] David Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology – CRYPTO'82.* Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 199–203.

[78] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1993, pp. 89–105.

[79] Hubert Comon-Lundh and Stéphanie Delaune. "The finite variant property: How to get rid of some algebraic properties". In: *RTA*. Vol. 3467. Springer. 2005, pp. 294–307.

[80] DL Cool and Angelos D Keromytis. "Conversion and proxy functions for symmetric key ciphers". In: *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*. Vol. 1. IEEE. 2005, pp. 662–667.

[81] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. "A survey of symbolic methods in computational analysis of cryptographic systems". In: *Journal of Automated Reasoning* 46.3-4 (2011), pp. 225–259.

[82] Ronald Cramer and Ivan Damgård. "Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge Be for Free?" In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 424–441.

[83] Cas J. F. Cremers. "Unbounded verification, falsification, and characterization of security protocols by pattern refinement". In: *ACM CCS 08: 15th Conference on Computer and Communications Security*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. Alexandria, Virginia, USA: ACM Press, Oct. 2008, pp. 119–128.

[84] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *Proceedings of ACM CCS 2017: 24th Conference on Computer and Communications Security*. 2017.

[85] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". In: *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 2016, pp. 470–485.

[86] Moti Yung, ed. *Advances in Cryptology – CRYPTO 2002*. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002.

[87] Reihaneh Safavi-Naini and Ran Canetti, eds. *Advances in Cryptology – CRYPTO 2012*. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012.

[88] Mario Di Raimondo and Rosario Gennaro. "Provably Secure Threshold Password-Authenticated Key Exchange". In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 507–523.

[89] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[90] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. "Authentication and Authenticated Key Exchanges". In: *Designs, Codes and Cryptography* 2.2 (June 1992), pp. 107–125.

[91] Yevgeniy Dodis. "Efficient Construction of (Distributed) Verifiable Random Functions". In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003, pp. 1–17.

[92] Yevgeniy Dodis and Aleksandr Yampolskiy. "A Verifiable Random Function with Short Proofs and Keys". In: *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Serge Vaudenay. Vol. 3386. Lecture Notes in Computer Science. Les Diablerets, Switzerland: Springer, Heidelberg, Germany, Jan. 2005, pp. 416–431.

[93] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[94] Danny Dolev and Andrew Chi-Chih Yao. "On the Security of Public Key Protocols (Extended Abstract)". In: *22nd Annual Symposium on Foundations of Computer Science*. Nashville, Tennessee: IEEE Computer Society Press, Oct. 1981, pp. 350–357.

[95] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. *A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol*. Cryptology ePrint Archive, Report 2016/081. `http://eprint.iacr.org/`. 2016.

[96] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. "A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015.* 2015, pp. 1197–1210.

[97] Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. "Beyond subterm-convergent equational theories in automated verification of stateful protocols". In: *International Conference on Principles of Security and Trust.* Springer. 2017, pp. 117–140.

[98] Paul Ducklin. *Anatomy of a password disaster – Adobe's giant-sized cryptographic blunder.* https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/. 2013.

[99] Thai Duong and Juliano Rizzo. *Here Come the $\oplus$ Ninjas.* Unpublished manuscript. May 2011.

[100] Thai Duong and Juliano Rizzo. *The CRIME Attack.* Ekoparty Security Conference presentation. 2012.

[101] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology – CRYPTO'84.* Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1984, pp. 10–18.

[102] Eli Biham, ed. *Advances in Cryptology – EUROCRYPT 2003.* Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003.

[103] Adam Everspaugh, Rahul Chatterjee, Samuel Scott, Ari Juels, Thomas Ristenpart, and Cornell Tech. "The Pythia PRF service". In: *Proceedings of the 24th USENIX Conference on Security Symposium.* USENIX Association. 2015, pp. 547–562.

[104] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. "Key Rotation for Authenticated Encryption". In: *Advances in Cryptology – CRYPTO 2017, Part III.* Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2017, pp. 98–129.

[105] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. “Key Confirmation in Key Exchange: A Formal Treatment and Implications for TLS 1.3”. In: *2016 IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 23-25, 2016*. 2016.

[106] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. “Random Oracles with(out) Programmability”. In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Dec. 2010, pp. 303–320.

[107] Warwick Ford and Burton S. Kaliski Jr. “Server-Assisted Generation of a Strong Secret from a Password”. In: *9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE 2000)*. Gaithersburg, MD, USA: IEEE Computer Society, June 2000, pp. 176–180.

[108] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. “Keyword Search and Oblivious Pseudorandom Functions”. In: *TCC 2005: 2nd Theory of Cryptography Conference*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 303–324.

[109] Christina Garman, Kenneth G Paterson, and Thyla Van der Merwe. “Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS.” In: *USENIX Security*. 2015, pp. 113–128.

[110] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. “Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography”. In: *17th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Brian A. Coan and Yehuda Afek. Puerto Vallarta, Mexico: Association for Computing Machinery, June 1998, pp. 101–111.

[111] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to Construct Random Functions”. In: *Journal of the ACM* 33.4 (Oct. 1986), pp. 792–807.

[112] Google. *Managing Data Encryption*. https://cloud.google.com/storage/docs/encryption.

[113] Marko Horvat. “Formal Analysis of Modern Security Protocols in Current Standards”. PhD thesis. University of Oxford, 2016.

[114] Jean-Marie Hullot. *A Catalogue of Canonical Term Rewriting Systems*. Tech. rep. 1980.

[115] Anca Ivan and Yevgeniy Dodis. “Proxy Cryptography Revisited”. In: *ISOC Network and Distributed System Security Symposium – NDSS 2003*. San Diego, CA, USA: The Internet Society, Feb. 2003.

[116] 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: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*. 2015, pp. 1185–1196.

[117] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. “Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model”. In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 233–253.

[118] Antoine Joux and Kim Nguyen. “Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups”. In: *Journal of Cryptology* 16.4 (Sept. 2003), pp. 239–247.

[119] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[120] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. “Attacking RSA-Based Sessions in SSL/TLS”. In: *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*. 2003, pp. 426–440.

[121] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. “(De-)Constructing TLS”. In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 20.

[122] Venkata Koppula, Kim Ramchen, and Brent Waters. “Separations in Circular Security for Arbitrary Length Key Cycles”. In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, Mar. 2015, pp. 378–400.

[123] Hugo Krawczyk. “Cryptographic Extraction and Key Derivation: The HKDF Scheme”. In: *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*. 2010, pp. 631–648.

[124] Hugo Krawczyk and Hoeteck Wee. "The OPTLS Protocol and TLS 1.3". In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 978.

[125] Russell W. F. Lai, Christoph Egger, Dominique Schröder, and Sherman S. M. Chow. "Phoenix: Rebirth of a Cryptographic Password-Hardening Service". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 899–916. ISBN: 978-1-931971-40-9.

[126] Brian A. LaMacchia, Kristin E. Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange". In: *Provable Security, First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007, Proceedings*. 2007, pp. 1–16.

[127] A. Langley and W. Chang. *QUIC Crypto*. Available at https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/. Google, June 2013.

[128] Adam Langley. *ed25519 for Go*. http://godoc.org/github.com/agl/ed25519.

[129] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. "On the Security of the Pre-shared Key Ciphersuites of TLS". In: *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. 2014, pp. 669–684.

[130] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 31–46.

[131] Isis Agora Lovecruft. *curve25519-dalek*. https://github.com/isislovecruft/curve25519-dalek/.

[132] Gavin Lowe. "A Hierarchy of Authentication Specifications". In: *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*. CSFW '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 31–. ISBN: 0-8186-7990-5.

[133] Anna Lysyanskaya. "Unique Signatures and Verifiable Random Functions from the DH-DDH Separation". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 597–612.

[134] Colm MacCárthaigh. *Security review of TLS1.3 0-RTT*. TLS mailing list post. Available at https://www.ietf.org/mail-archive/web/tls/current/msg23051.html. Internet Engineering Task Force, May 2017.

[135] Philip D. MacKenzie and Michael K. Reiter. "Delegation of Cryptographic Servers for Capture-Resilient Devices". In: *ACM CCS 01: 8th Conference on Computer and Communications Security*. Philadelphia, PA, USA: ACM Press, Nov. 2001, pp. 10–19.

[136] Philip D. MacKenzie and Michael K. Reiter. "Networked Cryptographic Devices Resilient to Capture". In: *2001 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE Computer Society Press, May 2001, pp. 12–25.

[137] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. "Threshold Password-Authenticated Key Exchange". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 385–400.

[138] Itsik Mantin. *Attacking SSL when using RC4*. White Paper. Mar. 2015.

[139] Nicholas D Matsakis and Felix S Klock II. "The rust language". In: *ACM SIGAda Ada Letters* 34.3 (2014), pp. 103–104.

[140] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology". In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 21–39.

[141] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. "A cross-protocol attack on the TLS protocol". In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 2012, pp. 62–72.

[142] Max Krohn and Chris Coyne. *Wrap Wallet*. https://keybase.io/warp.

[143] R. McMillan. "The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster". In: *Wired* (2014).

[144] Simon Meier. *Advancing automated security protocol verification*. Doctoral thesis, ETH Zurich, Switzerland. 2013.

[145] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. “Verifiable Random Functions”. In: *40th Annual Symposium on Foundations of Computer Science*. New York, NY, USA: IEEE Computer Society Press, Oct. 1999, pp. 120–130.

[146] Silvio Micali and Ray Sidney. “A Simple Method for Generating and Sharing Pseudo-Random Functions, with Applications to Clipper-like Escrow Systems”. In: *Advances in Cryptology – CRYPTO’95*. Ed. by Don Coppersmith. Vol. 963. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1995, pp. 185–196.

[147] Bodo Moeller. *Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures*. Unpublished manuscript. http://www.openssl.org/~bodo/tls-cbc.txt. May 2004.

[148] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE bites: Exploiting The SSL 3.0 Fallback*. Security Advisory. Google, Sept. 2014.

[149] Alec Muffet. *Facebook: Password Hashing & Authentication*. Presentation at Real World Crypto. 2015.

[150] Allec Muffet. *Facebook: Password Hashing and Authentication*. https://video.adm.ntnu.no/pres/54b660049af94.

[151] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. “Reconsidering Generic Composition”. In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 2014, pp. 257–274.

[152] Moni Naor, Benny Pinkas, and Omer Reingold. “Distributed Pseudo-random Functions and KDCs”. In: *Advances in Cryptology – EUROCRYPT’99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, May 1999, pp. 327–346.

[153] Tatsuaki Okamoto and David Pointcheval. “The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes”. In: *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Kwangjo Kim. Vol. 1992. Lecture Notes in Computer Science. Cheju Island, South Korea: Springer, Heidelberg, Germany, Feb. 2001, pp. 104–118.

[154] Kenneth G. Paterson and Thyla van der Merwe. “Reactive and Proactive Standardisation of TLS”. In: *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*. 2016, pp. 160–186.

[155] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. "Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Seoul, South Korea: Springer, Heidelberg, Germany, Dec. 2011, pp. 372–389.

[156] Kenneth G Paterson and Sriramkrishnan Srinivasan. "On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups". In: *Designs, Codes and Cryptography* 52.2 (2009).

[157] PCI Security Standards Council. "Requirements and Security Assessment Procedures". In: *PCI DSS v3.2*. 2016.

[158] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1992, pp. 129–140.

[159] Yvo Desmedt, ed. *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003.

[160] David Pointcheval and Jacques Stern. "Provably Secure Blind Signature Schemes". In: *Advances in Cryptology – ASIACRYPT'96*. Ed. by Kwangjo Kim and Tsutomu Matsumoto. Vol. 1163. Lecture Notes in Computer Science. Kyongju, Korea: Springer, Heidelberg, Germany, Nov. 1996, pp. 252–265.

[161] Andrei Popov. "TLS 1.3 Client Authentication". In: *Meeting proceedings of the IETF-93 Workshop, Prague*. Retrieved from https://www.ietf.org/proceedings/93/slides/slides-93-tls-2.pdf, 2015.

[162] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3 (draft, revision 10)*. Available at https://tools.ietf.org/html/draft-ietf-tls-tls13-10. Internet Engineering Task Force, Oct. 2015.

[163] Eric Rescorla. *TLS 1.3 specification pull request: Wip client auth revision #316*. https://github.com/tlswg/tls13-spec/pull/316/.

[164] Eric Rescorla and Brian Korver. *Guidelines for writing RFC text on security considerations*. RFC 3552 (Informational). Internet Engineering Task Force, July 2003.

[165] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. "Careful with Composition: Limitations of the Indifferentiability Framework". In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 487–506.

[166] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds". In: *ACM CCS 09: 16th Conference on Computer and Communications Security*. Ed. by Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis. Chicago, Illinois, USA: ACM Press, Nov. 2009, pp. 199–212.

[167] Phillip Rogaway. "Authenticated-Encryption With Associated-Data". In: *ACM CCS 02: 9th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri. Washington D.C., USA: ACM Press, Nov. 2002, pp. 98–107.

[168] Phillip Rogaway. "Formalizing Human Ignorance". In: *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*. Ed. by Phong Q. Nguyen. Vol. 4341. Lecture Notes in Computer Science. Springer, 2006, pp. 211–228. ISBN: 3-540-68799-8.

[169] Phillip Rogaway. "Practice-oriented provable security and the social construction of cryptography". In: *Unpublished essay* (2009).

[170] Phillip Rogaway and Thomas Shrimpton. "A Provable-Security Treatment of the Key-Wrap Problem". In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, Heidelberg, Germany, May 2006, pp. 373–390.

[171] Phillip Rogaway and Thomas Shrimpton. *Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem*. Cryptology ePrint Archive, Report 2006/221. http://eprint.iacr.org/2006/221. 2006.

[172] Jim Roskind. "QUIC: Multiplexed Stream Transport over UDP". In: *Google working design document* (2013).

[173] Amit Sahai, Hakan Seyalioglu, and Brent Waters. "Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 199–217.

[174] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. "Cryptosystems based on Pairing". In: *SCIS 2000*. Okinawa, Japan, Jan. 2000.

[175] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. Doctoral thesis, ETH Zurich, Switzerland. 2012.

[176] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. Ed. by Stephen Chong. IEEE, 2012, pp. 78–94.

[177] Jonas Schneider, Nils Fleischhacker, Dominique Schröder, and Michael Backes. "Efficient Cryptographic Password Hardening Services from Partially Oblivious Commitments". In: *ACM CCS 16: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. Vienna, Austria: ACM Press, Oct. 2016, pp. 1192–1203.

[178] Adi Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613.

[179] Thomas Shrimpton, Martijn Stam, and Bogdan Warinschi. "A Modular Treatment of Cryptographic APIs: The Symmetric-Key Case". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 277–307.

[180] *Tamarin prover GitHub repository (develop branch)*. https://github.com/tamarin-prover/tamarin-prover. 2015.

[181] University of Texas at Austin. Automatic Theorem Proving Project, DS Lankford, and AM Ballantyne. *Decision Procedures for Simple Equational Theories with Commutative Associative Axioms: Complete Sets of Commutative Associative Reductions*. 1977.

[182] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ..." In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. 2002, pp. 534–546.

[183] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. "Cross-VM side channels and their use to extract private keys". In: *ACM CCS 12: 19th Conference on Computer and Communications Security*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. Raleigh, NC, USA: ACM Press, Oct. 2012, pp. 305–316.