BERGISCHE
UNIVERSITÄT
WUPPERTAL

Doctoral Dissertation

On Real-World Cryptographic Protocols for End-to-End Encrypted Backups and Key Confirmation

Tobias Handirk, M.Sc.

August 23, 2024

Submitted to the
School of Electrical, Information and Media Engineering
University of Wuppertal

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

Tobias Handirk
Place of birth: Bad Oeynhausen, Germany
Author's contact information:
tobias.handirk@uni-wuppertal.de

Thesis Advisor:	**Prof. Dr.-Ing. Tibor Jager** University of Wuppertal, Wuppertal, Germany
Second Examiner:	**Prof. Dr. Kristian Gjøsteen** NTNU, Norwegian University of Science and Technology, Trondheim, Norway
Thesis submitted:	August 23, 2024
Thesis defense:	November 22, 2024
Last revision:	February 2, 2025

Acknowledgements

First and foremost, I want to thank Tibor Jager for giving me the opportunity to pursue a PhD and supervising me on this journey. It is a pleasure to work with you and I greatly enjoyed our discussions, especially when they led to yet another interesting problem in key exchange, even though you say you "never want to work on key exchange again."

Next, I want to thank Kai Gellert for inspiring me to follow this career path. Without you, I never would have thought about pursuing a PhD. Thank you for providing lots of guidance and almost being a second supervisor for me next to Tibor. You are an awesome collaborator, even when we suffered together through the deep depths of despair that come with trying to understand the UC framework.

Tibor and Kai, you are great persons and I learned many invaluable lessons from both of you. I cannot think of a better pair to lead this research group.

Thanks to all the other current and former, awesome colleagues in our group. In alphabetical order, thank you, Amin Faez, Christian Holler, David Niehues, Denis Diemert, Gareth T. Davies, Jan Drees, Jonas von der Heyden, Jutta Maerten, Lin Lyu, Marloes Venema, Máté Horváth, Pascal Bemmann, Peter Chvojka, Rafael Kurek, Raphael Heitjohann, Saqib Kakvi, and Tom Neuschulten. All of you create a great atmosphere in the group and made the time in Wuppertal an awesome experience. Special thanks to Marloes, Raphael, and Jan for being great office-mates.

Further, I want to say thank you to all my friends who accompanied me through school and university. Michi, Louisa, Nina, Caro, Malin, Theresa, Matze, and Lennart, I am thankful for having such great friends and that even though we do not see each other as often anymore as we used to, nothing ever changes. Fynn, Simon, Mike, and Viktor, thank you for making our time together in Paderborn an unforgettable one. All of you were a major part of the path that led me to this thesis.

There are too many people in all the football teams that I have been a part of to thank them all individually. As many of us like to say, football is the ultimate team sport, and I enjoyed being a part of all of them. To every Dolphin, Butcher, Panther, and Greyhound, thank you for creating moments that we will never forget and also contributing to this thesis by providing a great outlet. Leon and Thorben, you deserve a special thanks for all the late evenings after practice at our favorite restaurant.

Zuletzt möchte ich meiner Familie danken. Mama, Papa und Rebekka, vielen Dank, dass ihr mich bei allem unterstützt und immer an mich glaubt! Ohne euch wäre ich nicht bis hier gekommen.

Abstract

The widespread use of the Internet and mobile devices has sparked a rapid growth in digital communication and, in turn, made the use of cryptography an integral part of everyday life, as countless applications are only enabled through the protection of confidentiality, integrity, and authenticity of sensitive data. In this thesis, we focus on two particular aspects of cryptography in the real world.

In the first part, we focus on authenticated key exchange (AKE), which is a fundamental building block for many cryptographic protocols deployed in practice. More specifically, we concentrate on the concrete security of the well-known key confirmation paradigm. Adding key confirmation to an AKE protocol crucially enhances the security of the protocol, upgrading it from weak to full forward security and from implicit to explicit authentication. In this thesis, we show that for a large and natural class of protocols, this upgrade comes at the cost of non-tight security proof. More precisely, we propose a novel meta-reduction technique, which we employ to prove that almost all key confirmation protocols must have at least a linear security loss in the number of users. Additionally, we construct a highly efficient key confirmation protocol with a linear security loss, yielding AKE protocols with full forward security and *optimal* tightness.

In the second part, we turn our attention to instant messaging, a highly relevant application, where end-to-end encryption (E2EE) has become the de facto security standard. While the security of the data in transit has received a lot of attention, the security of the data at rest has so far been somewhat neglected. In this thesis, we conduct the first formal security analysis of the E2EE backup protocol deployed by WhatsApp, which is built on top of the OPAQUE password authenticated key exchange. We show that, by relying on a hardware security module (HSM), the protocol provides strong security guarantees for the users' passwords and chat histories even against a maliciously acting WhatsApp server.

Finally, we explore how we can design E2EE backup protocols with a tradeoff between simplicity and efficiency of protocols on the one hand and minimal trust assumptions on the other hand. We propose three different protocols that lie on a spectrum reaching from fully trusting an HSM, assuming that it is completely incorruptible, to not trusting the HSM at all and assuming that all data and cryptographic keys maintained by the HSM may be corrupted. For each scenario we aim to give *minimal* solutions, achieving a simple yet efficient protocol design.

Contents

Acronyms

AE authenticated encryption

AEAD authenticated encryption with associated data

AKE authenticated key exchange

aPAKE asymmetric password-based key exchange protocol

DH Diffie–Hellman

E2EE end-to-end encryption (or end-to-end encrypted)

HSM hardware security module

IM instant messaging

MAC message authentication code

OPRF oblivious pseudorandom function

PKE public-key encryption

PPKR password-protected key retrieval

PPSS password-protected secret sharing

PPT probabilistic polynomial time

ROM random oracle model

SE symmetric encryption

TLS Transport Layer Security

UC universal composability

WBP WhatsApp backup protocol

1 Introduction

Cryptography is ever present in today's society, having become a cornerstone of digital communication. According to Statista, the number of worldwide Internet users as of July 2024 is around 5.5 billion,[1] constituting roughly two-thirds of the entire global population. As the number of individuals utilizing the Internet continues to grow, the amount of private and sensitive data transmitted over the Internet on a daily basis rises steadily. In 2017, Cisco evaluated the global amount of daily Internet traffic to be around 4 Exabytes, equivalent to 4 000 000 Terabytes. Furthermore, they estimated that the Internet traffic would more than triple within five years and would reach more than 13 Exabytes per day in 2023.[2] Given that the Internet is inherently an insecure network where adversaries can eavesdrop, intercept, and modify exchanged data, protecting these vast amounts of private data against unauthorized access is critical.

In order to enable the exchange of data over insecure networks, cryptographic techniques must be employed to safeguard the confidentiality, authenticity, and integrity of the exchanged data. As such, cryptography is a vital tool for countless everyday-use applications. To illustrate, consider the case of online banking, where the confidentiality of login data, as well as the integrity and authenticity of financial transactions, must be ensured. Similarly, contactless payment systems—which according to the European Central Bank already made up approximately 75% of all non-cash payments in 2022[3]—require that the customer's device, which may be a bank card, smartphone, or wearable, and the terminal accepting the payment authenticate towards each other to prevent, for example, payment fraud. As a final example, let us highlight instant messaging apps, which have a total monthly active user base exceeding 6.6 billion.[4] Instant messaging apps are used not only by private individuals but also by companies to communicate with customers and business partners, requiring their conversations to be kept private to protect their businesses.

These data and statistics emphasize the significant global demand for protection against unauthorized access to transmitted data. In the absence of cryptography, the aforementioned applications, in addition to numerous other applications, would not exist, and people would lose access to many technologies that they have

[1] See `https://www.statista.com/statistics/617136/digital-population-worldwide/`

[2] See `https://cloud.report/Resources/Whitepapers/eea79d9b-9fe3-4018-86c6-3d1df813d3b8_white-paper-c11-741490.pdf`

[3] See `https://data.ecb.europa.eu/blog/blog-posts/contactless-payments-euro-area`

[4] See `https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/`

become accustomed to.

1.1 Key Confirmation in Authenticated Key Exchange

One of the essential cryptographic mechanisms in this landscape is *authenticated key exchange* (AKE), which allows two parties to establish a shared secret over an insecure channel [BR94]. Subsequently, the shared secret is used in other cryptographic building blocks to ensure security properties such as confidentiality while exchanging the data. In this way, AKE forms the basis for securing many of the above applications.

The most prevalent AKE protocol in practice is the *Transport Layer Security* (TLS) protocol [Res18]. Its most recent version 1.3 was released in 2018 and is used to secure many application protocols, such as HTTP [TB22], IMAP [ML21], and SMTP [Kle08]. Approximately 80% of all webpages accessed with the Mozilla Firefox browser are secured via TLS[5], highlighting the practical relevance of AKE protocols.

Security of AKE. The fundamental security property of AKE protocols is that the shared secrets established by the protocol appear to be random to any adversary. A more advanced security property is *forward security*, which refers to the ability of an AKE protocol to maintain the apparent randomness of previously established shared secrets even after an adversary compromises the long-term secret key of some party [Gün90]. Given the prevalence of threats such as compelled access by law enforcement, global surveillance programs as exposed by the Snowden revelations, and vulnerabilities in software implementations, which all could allow an adversary to compromise secret keys, forward security is considered a standard security goal of modern cryptography.

Forward security essentially comes in two different flavors. For *weak* forward security, we only consider passive adversaries who merely observe the protocol messages, while for *full* forward security, we consider active adversaries who may actively interfere with the messages. Full forward security clearly provides stronger security properties, making it the preferred version for protocols deployed in practice. TLS 1.3 as the most common AKE protocol in practice indeed provides full forward security [Die23].

Key Confirmation. It is naturally easier to develop AKE protocols achieving only weak forward security instead of full forward security since they do not need to provide as strong security guarantees. Therefore, a common approach for the design of fully secure protocols is initially to construct a weakly secure protocol and then apply a generic transformation that upgrades it to full forward security

[5] `https://letsencrypt.org/stats/`

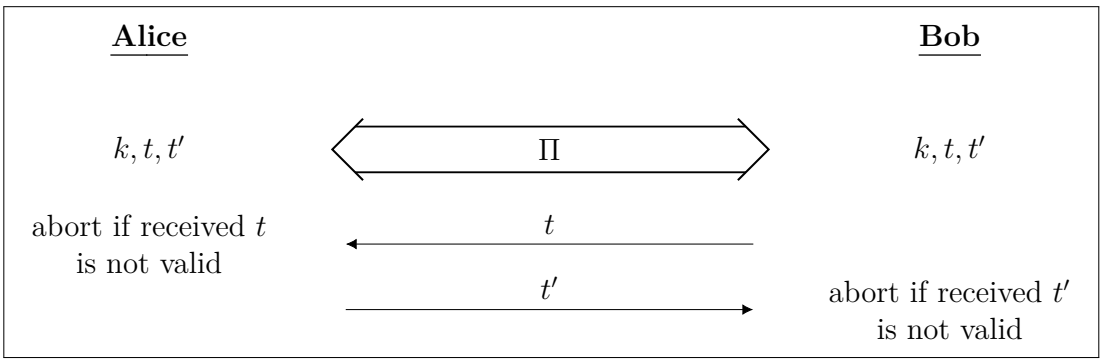

Figure 1.1: Visualization of the key confirmation paradigm. Alice and Bob execute the weakly secure protocol Π, deriving a shared secret k and two additional values t and t', which they exchange to confirm that they computed the same shared secret k.

[BWJM97, BWM99, Kra05]. A well-studied approach for this is the *key confirmation* paradigm [BPR00, Kra05, Yan13a, FGSW16, CCG+19, DFW20, GGJJ23], which is used in several protocols, such as the STS protocol [DvW92], different Oakley modes [Orm98], or SKEME [Kra96], cf. also [BMS20]. When applying the key confirmation paradigm, the parties derive additional values from the weakly secure protocol besides the shared secret itself and exchange these values in additional key confirmation messages, which essentially serve as a confirmation that both parties computed the same key. We visualize this process in Figure 1.1.

Now, actively interfering with the messages of the weakly secure protocol is of no advantage for the adversary since this would lead to the parties deriving different key confirmation messages and aborting the protocol. This means the adversary would have to interfere with the key confirmation messages as well and compute the correct messages such that the parties remain oblivious to the interference; however this would require knowledge of the long-term secret key already at the time of protocol execution. In this way, the key confirmation paradigm essentially reduces active adversaries to being passive, thereby carrying the weak forward security guarantees of the underlying protocol over to full forward security for the extended protocol.

Concrete Security of Key Confirmation. Although key confirmation is a well-known paradigm and has been deployed in several protocols, its *concrete security* has not yet been studied much. Concrete security [BR09] aims to obtain exact security bounds for cryptographic schemes. More specifically, the goal is to relate in precise, quantitative terms how difficult it is to break a scheme relative to some underlying mathematical problem. Informally, a cryptographic scheme is said to have *tight* security if the probability of breaking its security is the same as the probability of solving the underlying problem up to a small constant factor. On

the other hand, a scheme has *non-tight* security if these probabilities differ by a non-constant, polynomial factor.

To instantiate the parameters of a cryptographic protocol in a theoretically sound way, we need to consider whether it has tight or non-tight security and potentially compensate with larger parameter choices in the latter case. As larger parameters usually directly translate to an increased computational load, tight security is highly desired. For an illustration of the impact that non-tight security proofs can have, we refer the reader to the work by Davis and Günther [DG21], who compare the concrete security of TLS 1.3 for tight and non-tight proofs. This leads us to the central question for Part I of this thesis:

Is it possible to construct simple and efficient key confirmation protocols with tight security?

Our contributions, Part I. In the first part of this thesis we make the following contributions to explore the above question:

- We demostrate that a key confirmation protocol given by Cohn-Gordon *et al.* [CCG+19] that the authors claimed to be tightly secure is actually non-tigth by revealing a bug in their security analysis.
- We develop a simple, yet highly efficient (albeit non-tight) key confirmation protocol and prove its security.
- We show that a large class of protocols are inherently unable to achieve tight security, proving that no protocol with tighter security than our protocol can exist.

1.2 End-to-End Encrypted Backups

In the second part of this thesis, we turn to a prominent application that is built from AKE, namely *instant messaging* (IM) apps, which are some of the most popular applications on the Internet with billions of active users[6] and hundreds of billions of messages delivered per day.[7] They usually deploy (a generalized variant of) AKE to achieve *end-to-end encryption* (E2EE), meaning that the confidentiality, authenticity, and integrity of exchanged messages are provided even in the event that the IM service provider is compromised [CCD+17, BSJ+17, RMS18, ACD19, JMM19, CPZ20, VGIK20, BFG+22, CJSV22]. The widespread adoption of E2EE as the standard security goal for IM following the Snowden revelations in 2013 marks one of the most significant advancements for security on the Internet. However, the rise of E2EE also brought up new challenges.

[6] See `https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/`

[7] See `https://x.com/wcathcart/status/1321949078381453314`

Bypassing E2EE. While the security of the E2EE messaging protocols of several IM apps has been analyzed in many works, these analyses focus on the *transmission* of messages. However, these analyses alone are not sufficient for a convincing claim of achieving E2EE, as this requires all other features offered by the given app to be secure as well. A nearly ubiquitous feature among IM apps is the ability to back up chat histories to allow the recovery of the chats if the device is lost or the user switches to a new phone.[8] This indeed opens up opportunities for adversaries to potentially bypass the E2EE of the transmission and instead attack the backed up data. Therefore, the same strong security guarantees as for the transmission protocol are required from the backup protocols.

A natural approach to developing a backup protocol, which is for example deployed by WhatsApp, is to let the client generate a random key and simply encrypt the chat history under this key before storing the resulting ciphertext in some cloud storage. In order to allow the recovery of the backup in case the phone is lost, the key is stored by the IM service provider and retrieved by the client whenever necessary. Unfortunately, this simple protocol clearly fails to provide strong security guarantees, as a compromise of the cloud storage and WhatsApp's key storage would straightforwardly allow an adversary to decrypt the backup. In fact, this weakness was seemingly exploited recently, where previously well protected private communication became evidence in a lawsuit [Nov18].

E2EE Backups from Secure Hardware. Protocols aiming at the secure backup of a user's chat history—or a user's cryptographic key in general—have been proposed by Signal [Lun19], Apple [Krs16], Google [Wal18] and WhatApp [Wha21a].[9] The common idea in all approaches is to back up the user's high-entropy (encryption) key on a server and enable retrieval when the user correctly authenticates via a human-memorable secret, like a password or PIN. In particular, when the backup service is offered by the same provider that handles the E2EE communication, extra care is necessary to not undermine the encryption security. This is done by relying on trusted hardware enclaves, such as *hardware security module*s (HSMs). Intuitively, an HSM can be programmed once with software and then "locked" in such a way that it is infeasible to change its software afterwards. This enables even the protection of a protocol against corruption of the party running the HSM. The challenge now lies in designing the code run by the HSM in such a way that (1) users can retrieve their backup keys from a password or PIN, but (2) the IM service provider—not knowing the user password—cannot obtain the backup key.

[8] See e.g. `https://www.facebook.com/help/messenger-app/677912386869109?helpref=faq_content&locale=en_US`, or `https://threema.ch/en/faq/data_backup`, or `https://faq.whatsapp.com/481135090640375/?locale=en_US&cms_platform=android&cms_id=481135090640375&draft=false`

[9] The E2EE backups in WhatsApp notably are an *opt-in* feature, leaving the weaker protocol as the default option.

The WhatsApp E2EE Backup Protocol. Of the aforementioned approaches, the *WhatsApp backup protocol* (WBP) has enjoyed the most attention due to its enhanced protocol design and widespread usage. By early 2023, over 100 million WhatsApp users had already switched to this option [Cat22]. The WBP deploys OPAQUE [JKX18], an *asymmetric password-based key exchange protocol* (aPAKE) [GMR06] that allows a key exchange from a password without disclosing the password itself to the server. Furthermore, the WBP aims to provide security against *password guessing attacks*, where a malicious client repeatedly executes the retrieval protocol with password guesses pw'. If the password guess pw' equals the password pw used during initialization, an adversary would gain access to the secret backup key. Note that this attack is especially effective if the user password only has low entropy, which is often the case for human-memorable passwords in practice. The deployed protocol limits the number of admissible incorrect guesses to ten [Wha21a, DLS21], after which the HSM destroys the encrypted version of the backup key (and thus makes the backup irrecoverable). This guarantee should even hold if the WhatsApp server were to be compromised. This leads us to our first central question for Part II of this thesis:

Which formal security properties do we expect from an end-to-end encrypted backup protocol and are they all achieved by WhatsApp's backup protocol?

The Role of the HSM. The WBP protocol crucially relies on an HSM that is run within the server as an incorruptible entity. Assuming that all data and keys stored on the HSM are incorruptible can make the security analysis considerably less complex. While this captures the initial design choices made by the WBP protocol and the security claims WhatsApp advertises for its protocol [Wha21a], this is somewhat unsatisfactory from a security and protocol design perspective. First, if an incorruptible HSM can be assumed, the protocol could take more advantage of that—the core of the WBP protocol is the OPAQUE protocol that provides strong security guarantees even when the cryptographic state gets compromised, which wouldn't be needed if the assumption is that such an event can never occur. Second, relying on a perfectly secure sub-entity is a risky assumption. In fact, also trusted hardware modules have a history of getting breached or having to lower their security claims [VBMW+18, VBPS17, BC19, SRW22]. Thus, it would be desirable to clearly express and analyze the impact a partial or full corruption of the HSM has on the expected security guarantees of the backup protocol. This raises the final central question of this thesis:

How do we model security under partial or full corruption of the HSM and can we construct simple and efficient end-to-end encrypted backup protocols secure under the different corruption scenarios?

Our contributions, Part II. Our contributions in the second part of this thesis are the following.

- We introduce a novel cryptographic primitive that abstracts the properties of an E2EE backup protocol and formalize the expected security guarantees including a fine-grained HSM corruption model.
- We provide the first full description of the WBP and conduct an extensive security analysis of the WBP, revealing an attack where a corrupt WhatsApp server could get more than ten password guesses.
- We develop three novel protocols for E2EE backups, each tailored to a different corruption scenario of the HSM, and prove their security.

1.3 Publication Overview

This thesis is based on the following works. We provide details on the contributions of the author of this thesis at the beginning of each chapter.

Peer-reviewed publications:

[DFG+23] Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. Security analysis of the WhatsApp end-to-end encrypted backup protocol. In Helena Handschuh and Anna Lysyanskaya, editors, CRYPTO 2023, Part IV, volume 14084 of LNCS, pages 330–361, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[FHH+24] Sebastian H. Faller, Tobias Handirk, Julia Hesse, Máté Horváth, and Anja Lehmann. Password-protected key retrieval with(out) HSM protection. In ACM CCS 2024, Salt Lake City, Utah, USA, October 14–18, 2024. ACM Press. To appear.

Unpublished work:

[GGH+23] Kai Gellert, Kristian Gjøsteen, Tobias Handirk, Håkon Jacobsen, and Tibor Jager. On the concrete security of key confirmation. Work in progress.

Other Publications not included in the thesis:

[DDG+20] Fynn Dallmeier, Jan P. Drees, Kai Gellert, Tobias Handirk, Tibor Jager, Jonas Klauke, Simon Nachtigall, Timo Renzelmann, and Rudi Wolf. Forward-secure 0-RTT goes live: Implementation and performance analysis in QUIC. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, CANS 20, volume 12579 of LNCS, pages 211–231, Vienna, Austria, December 14–16, 2020. Springer, Cham, Switzerland.

[GH21] Kai Gellert and Tobias Handirk. A formal security analysis of session resumption across hostnames. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, ESORICS 2021, Part I, volume 12972 of LNCS, pages 44–64, Darmstadt, Germany, October 4–8, 2021. Springer, Cham, Switzerland.

[LGM+20] Sebastian Lauer, Kai Gellert, Robert Merget, Tobias Handirk, and Jörg Schwenk. T0RTT: Non-interactive immediate forward-secret single-pass circuit construction. PoPETs, 2020(2):336–357, April 2020.

2 Preliminaries

In this chapter, we introduce the notation used throughout this thesis. Further, we recap some fundamental concepts of cryptography as well as some standard cryptographic building blocks and computational problems.

2.1 Notation

We denote the security parameter as λ. For any $n \in \mathbb{N}$ let 1^n be the unary representation of n. We denote the length of a string s as $|s|$ and the concatenation of two strings a, b as $a \| b$. We define the binary operator $\stackrel{?}{=}$ to return true if the two operands are equal and false otherwise. We write $x \xleftarrow{\$} S$ to indicate that we choose an element x uniformly at random from set S and $|S|$ to indicate the size of S. For a *probabilistic polynomial time* (PPT) algorithm $\mathcal{A}$ we define $y \xleftarrow{\$} \mathcal{A}(x_1, \ldots, x_\ell)$ as the execution of $\mathcal{A}$ (with fresh random coins) on input $x_1, \ldots, x_\ell$ and assigning the output to y. We say an algorithm is *efficient* if it is a PPT algorithm.

If a scheme consists of a tuple of algorithms, such as $\mathsf{Scheme} = (\mathsf{Alg}_1, \mathsf{Alg}_2)$, we write $\mathsf{Scheme}.\mathsf{Alg}_1$ to refer to the algorithm Alg_1 of scheme Scheme. We instead sometimes only write Alg_1 if due to the context it is easy to see which scheme Alg_1 refers to (e.g. due to uniqueness of name).

We use records of form $\langle x_1, x_2, x_3 \rangle$ for bookkeeping in some of our formal arguments. For convenience, we introduce a notation that combines retrieval and assignment of such records, i.e., when retrieving $\langle \mathsf{value}, *, * \rangle$, we retrieve a record that contains the value value in the first field and arbitrary values in the second and third field (denoted by a wildcard symbol $*$). Additionally, we use brackets to indicate variable assignment after retrieval, i.e., when retrieving $\langle \mathsf{value}, [x_2], [x_3] \rangle$, we retrieve the record holding the value value in its first entry and assign the second and third entry to the variables x_2 and x_3, respectively.

2.2 Provable Security

The concept of provable security forms the backbone of modern cryptography. It was first introduced in the seminal work by Goldwasser and Micali in 1984 [GM84] and allows us to precisely understand the security guarantees that cryptographic schemes or protocols provide. The basic idea is to first define a security model that formally describes what it means for an adversary to "break the security" of a given cryptosystem. Then, a rigorous mathematical proof is developed that relates

the security of the cryptosystem to a computational problem that is assumed to be hard, e.g. the discrete logarithm or factorization of integers. This shows that the cryptosystem is indeed secure if no algorithm that efficiently solves the computational problem exists.

Nowadays, several approaches to implementing this general paradigm exist, including *game-based security* as proposed by Goldwasser and Micali [GM84], the *universal composability* (UC) framework due to Canetti [Can01], the *reactive simulatability* framework developed by Backes, Pfitzmann, and Waidner [BPW07], and the *constructive cryptography* framework introduced Maurer [Mau11]. In this thesis, we use both game-based security and the UC framework, and in the following provide a brief introduction to both approaches.

2.2.1 Game-based Security

In the game-based approach, the security model is a *security game*, also called *security experiment*, played between a challenger and the adversary $\mathcal{A}$. We denote security games as $\mathbf{Exp}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}}$, meaning that the experiment implements the security goal or security property Goal for the scheme Scheme. In the experiment, the challenger sets up a well-defined problem or challenge for the adversary, where solving the challenge reflects breaking the desired security goal. This challenge could for example be the recovery of an encrypted message, where the adversary does not know the key under which the message was encrypted.

There are essentially two types of challenges: search problems and decision problems. In a search problem, the adversary is tasked with outputting a string of arbitrary form, such as the message in the aforementioned challenge of message recovery. In a decision problem, it is tasked with distinguishing between two possible situations, which for example could be determining which one of two messages chosen by the adversary was encrypted by the challenger. At the end of the game, the challenger always outputs either 0 or 1, indicating whether the adversary was successful in solving the challenge.

Finally, we consider the *advantage* $\mathbf{Adv}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}}$ that measures the probability of the adversary winning the experiment. For search problems this is simply defined as the probability of the challenger outputting 1, i.e., we have $\mathbf{Adv}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}} := \Pr[\mathbf{Exp}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}} = 1]$. But for decision problems we need to account for the fact that the adversary can always guess correctly with probability $1/2$ as it only needs to distinguish between two equally likely possibilities. Hence, for decision problems we always define $\mathbf{Adv}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}} := \left|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}} = 1] - 1\right|$. Using this formalism, we now have the tools to describe when a scheme is considered secure. There are two common approaches for this, namely the *asymptotic* and the *concrete* approach.

The Asymptotic Approach. In the asymptotic approach [GM84], the scheme as well as the adversary are parameterized by a *security parameter* $\lambda \in \mathbb{N}$, which

can typically be viewed as corresponding to the length of the key. Then, a scheme is considered secure if for any efficient adversary its advantage is asymptotically small in λ. More precisely, we use the following notion of *negligible* functions.

Definition 1. A function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$, where $\mathbb{R}^{\geq 0}$ denotes the non-negative real numbers, is *negligible in* n if for every polynomial p there is an $N \in \mathbb{N}$ such that for $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

We say a scheme is Goal-secure if for all adversaries with runtime polynomial in λ its advantage is negligible in λ. Informally speaking, this can also be interpreted as "there exist sufficiently large values of λ such that the probability of an adversary breaking the security is vanishingly small."

The Concrete Approach. In contrast to the asymptotic approach, the concrete approach [BR09] aims to provide explicit bounds on the runtime and success probability of the adversary. This captures what practitioners are usually interested in, as it gives exact probabilities of the scheme being secure for the parameters chosen for deployment. Here, we say a scheme is (t, ϵ)-Goal-secure if any adversary with runtime at most t has an advantage of at most ϵ.

Security Proofs. Having established how we model security in the game-based setting, we now come back to how we prove the security of a given scheme. As mentioned before, we want to relate the security of the scheme to a computational problem with the goal of deriving statements of the form "if the problem Problem is hard, then the scheme Scheme is Goal-secure."[1] As such a statement is usually difficult to prove, we instead consider the equivalent statement "if the scheme Scheme is not Goal-secure, then the problem Problem is not hard." To prove this, we assume that an efficient adversary $\mathcal{A}$ exists that breaks the Goal-security of Scheme. The central part of the proof is then the construction of an efficient *reduction* $\mathcal{R}$ that plays the experiment $\mathbf{Exp}^{\mathsf{Problem}}_{\mathcal{R}}$ and at the same time runs $\mathcal{A}$ as a subroutine and simulates the experiment $\mathbf{Exp}^{\mathsf{Goal}}_{\mathsf{Scheme},\mathcal{A}}$ for $\mathcal{A}$. If the reduction is able to transform the output of the adversary in the simulated experiment into a solution to the problem Problem, we have successfully shown that Scheme is Goal-secure if Problem is hard.

Tightness of Security Proofs and Theoretically Sound Instantiations. Especially when using the concrete approach, we are often interested in the *tightness* of a reduction or its *security loss*. The security loss can be defined as an upper bound on the factor ℓ such that

$$t_{\mathcal{R}}/\epsilon_{\mathcal{R}} \leq \ell \cdot t_{\mathcal{A}}/\epsilon_{\mathcal{A}},$$

[1] Note that often we do not show that Scheme is Goal-secure if some computational problem is hard, but instead show that Scheme is Goal-secure if some scheme Scheme′ is Goal′-secure.

where $t_{\mathcal{R}}$ and $t_{\mathcal{A}}$ are the running times of the reduction $\mathcal{R}$ and the adversary $\mathcal{A}$, respectively, and $\epsilon_{\mathcal{R}}$ and $\epsilon_{\mathcal{A}}$ are their advantages. A reduction is said to be *tight*, if ℓ is a small constant. It is *non-tight*, or *lossy*, if ℓ depends on the adversary or on deployment parameters, such as the number of users.

Considering the tightness of a security proof allows us to relate the security of some cryptographic scheme to some computational problem in precise, quantitative terms. Moreover, we can ideally instantiate the scheme in a theoretically sound way. To this end, we consider the security loss of the reduction and can compensate for a lossy reduction with matching parameters, such as larger algebraic groups or key lengths. A larger security loss requires larger cryptographic parameters for a theoretically sound instantiation, which in turn may incur significant overhead in terms of computations and communication. In particular, if the concrete security bound depends on the application context, such as the number of users, then this must be adequately compensated. This can be difficult if the exact number is not known at the time of the deployment. In this case, upper bounds must be determined, which may incur significant and possibly unnecessary computational overhead. A protocol with tight security can be instantiated with optimal parameters in a theoretically sound way, independent of the concrete (and possibly *a priori* unknown) application context.

2.2.2 Universal Composability

In the UC framework, security is modeled via *ideal functionalities*. For a given scheme or protocol that we want to analyze, an ideal functionality is an abstraction of the cryptographic primitive that describes what parties should ideally learn when interacting with the primitive. An ideal functionality can often be thought of as a trusted third party, which takes the inputs of all parties, executes the protocol internally, and hands the computed output back to the parties.

The basic idea of proving a protocol secure in the UC framework is to describe a *simulator* that "mimics" the behavior of the real protocol towards a distinguishing entity called the *environment*. The environment represents the external context in which the protocol runs, including other protocols, or human user and adversarial interaction. Hence, the environment may interactively instruct any party to execute any part of the protocol with inputs of its choosing, intercept and modify messages between parties, and even corrupt parties such that it can let them run arbitrary code.

A protocol π is considered secure with respect to some ideal functionality if no efficient environment is able to distinguish with non-negligible probability whether it interacts with parties running the real protocol or with the simulator Sim that is mimicking the protocol together with the ideal functionality $\mathcal{F}$. Formally, we define the advantage of an environment $\mathcal{Z}$ as

$$\mathbf{Adv}^{\mathcal{F}}_{\pi,\textsc{Sim},\mathcal{Z}}(\lambda) := \left|\Pr[\textsc{Exec}_{\pi,\mathcal{Z}}(\lambda) = 1] - \Pr[\textsc{Ideal}_{\mathcal{F},\textsc{Sim},\mathcal{Z}}(\lambda) = 1]\right|,$$

where $\text{EXEC}_{\pi,\mathcal{Z}}$ denotes the execution of the protocol π interacting with environment $\mathcal{Z}$, and $\text{IDEAL}_{\mathcal{F},\text{SIM},\mathcal{Z}}$ denotes the execution of the "ideal" protocol simulated by SIM with $\mathcal{F}$ while interacting with $\mathcal{Z}$. A mathematically precise definition of these executions is beyond the scope of this thesis and we refer to [Can01].

We say π is secure with respect to $\mathcal{F}$ if for any efficient environment $\mathcal{Z}$ its advantage is negligible in λ. In UC terminology we also say the protocol UC-realizes the ideal functionality. Proving that the two view are indistinguishable often involves a similar approach as the security proofs in the game-based setting. We often assume that an environment exists that is able to distinguish between the two views and then construct a reduction that solves some computational problem or breaks some security property of a cryptographic primitive used in the protocol.[2] If the given computational problem is indeed hard or we have previously proven the given security property of the given primitive, this shows that the two views are indeed indistinguishable.

Informally speaking, proving that a protocol UC-realizes some functionality can be interpreted as showing that no party including an adversary can learn more from interacting with the real protocol than what they learn from the ideal functionality, yielding a lower bound on the security guarantees of the protocol. This also means that even though we can think of functionalities as trusted third parties — which would not leak anything to an adversary — in order to prove protocols secure, functionalities must often leak some information to the adversary. As a toy example, imagine a simple scenario where Alice just wants to send a secret message to Bob, for which they would use some symmetric encryption scheme. Then, by simply observing the ciphertext sent by Alice, the adversary can learn the length of the message. Thus, an ideal functionality for this scenario would also have to leak this information, as we cannot prove the protocol secure with respect to a functionality that does not leak the length of the message to the adversary.

This highlights the importance of defining meaningful functionalities in the UC framework. On one hand, the functionality must leak enough information such that we are able to prove protocols secure, but on the other hand it should leak as little as possible since we obviously desire strong security guarantees. A challenge in designing ideal functionalities is hence finding the sweet spot where the functionality provides only the leakage that is necessary to allow a simulator to simulate the protocol but no other leakage.

A key feature of the UC framework is the composition theorem. Imagine a protocol that uses some cryptographic primitive, e.g. symmetric encryption, as a building block. The composition theorem then states that any instantiation of the primitive that is secure in the UC framework remains secure when it is used as the building block in the protocol, even if it is used concurrently in parallel multiple times. This allows for a modular design of complex cryptographic protocols combining smaller components into larger systems. A similar composition does

[2]Note that constructing this reduction does not require the security property to be formalized in the UC framework and can also be done with game-based notions for the security property.

not hold in general for protocols proven secure in game-based security models (see e.g. [Sho99]).

2.3 Cryptographic Building Blocks

In this section, we recap the syntax and security of some basic cryptographic primitives. For a more extensive discussion we refer the reader to [KL21]. In the following, we use the game-based approach with asymptotic security, although all security definitions can be formulated in the concrete security setting or the UC framework as well (with the exception of cryptographic hash functions).

2.3.1 Cryptographic Hash Functions

A *hash function* is a function that takes inputs of arbitrary length and maps them to outputs of a fixed length. In cryptography, intuitively the security property we require from hash functions is that it is difficult to find two inputs $x \neq x'$ that are mapped to the same output, which is called a *collision*. However, since hash functions $H : \{0,1\}^* \to \{0,1\}^\lambda$ are compressing functions in the sense that their range is larger than their domain, such collisions must always exist. Thus, for any given hash function, there always exists an efficient adversary that outputs a collision with non-negligible probability, e.g. the adversary that has the collision hard-coded as its output. For this reason, we follow the approach by Rogaway [Rog06] and say a hash function is collision resistant if we cannot construct an adversary that finds a collision with non-negligible probability.

Definition 2. A hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ is *collision resistant* if we cannot efficiently construct an efficient adversary $\mathcal{A}$ whose advantage

$$\mathbf{Adv}^{\mathsf{CollRes}}_{H,\mathcal{A}}(\lambda) \coloneqq \Pr[x \neq x' \wedge H(x) = H(x') \mid (x, x') \xleftarrow{\$} \mathcal{A}(1^\lambda)]$$

is non-negligible.

2.3.2 Symmetric Encryption

A *symmetric encryption* (SE) scheme allows private communication between parties sharing some private key k. A message m can be encrypted under k to obtain a ciphertext c, which can be decrypted using the same key to obtain m.

Definition 3. A *symmetric encryption scheme* consists of three PPT algorithms $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$.

$\mathsf{KGen}(1^\lambda)$**:** takes as input a security parameter λ and outputs a key $k \in \mathcal{K}$.

$\mathsf{Enc}(k, m)$**:** takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.

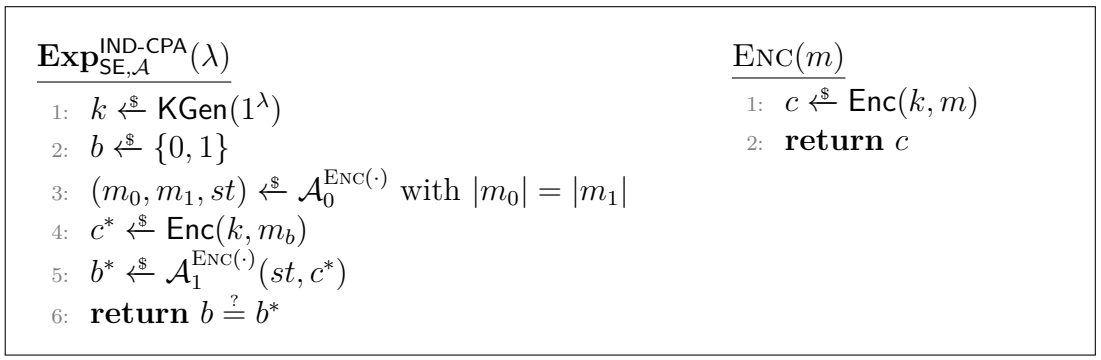

Figure 2.1: Security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ for IND-CPA-security of symmetric encryption.

$\mathsf{Dec}(k, c)$**:** takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$ or an error symbol $\perp$.

We say a scheme SE is *correct* if for all $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ and all $m \in \mathcal{M}$ it holds that

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m.$$

For security, we intuitively require that the ciphertext c leaks no information about the message m. This is formalized with the concept of *indistinguishability*. Here the adversary chooses two messages m_0, m_1 of the same length and gets the encryption of either m_0 or m_1 under a randomly chosen key. It then has to distinguish which of the two messages was encrypted. More precisely, we consider two standard security notions, namely *indistinguishability under chosen plaintext attacks* (IND-CPA) and *indistinguishability under chosen ciphertext attacks* (IND-CCA), which additionally give the adversary access to an encryption oracle, resp. an encryption and a decryption oracle. The formal security experiments are given in Figures 2.1 and 2.2.

Definition 4. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *indistinguishability of ciphertexts under chosen plaintext attacks* (IND-CPA) of an SE scheme $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{A}}(\lambda) := \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{A}}(\lambda) = 1\right] - 1 \right|.$$

We say SE is IND-CPA-secure if $\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

Definition 5. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *indistinguishability of ciphertexts under chosen ciphertext attacks* (IND-CCA) of an SE scheme $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{SE},\mathcal{A}}(\lambda) := \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{SE},\mathcal{A}}(\lambda) = 1\right] - 1 \right|.$$

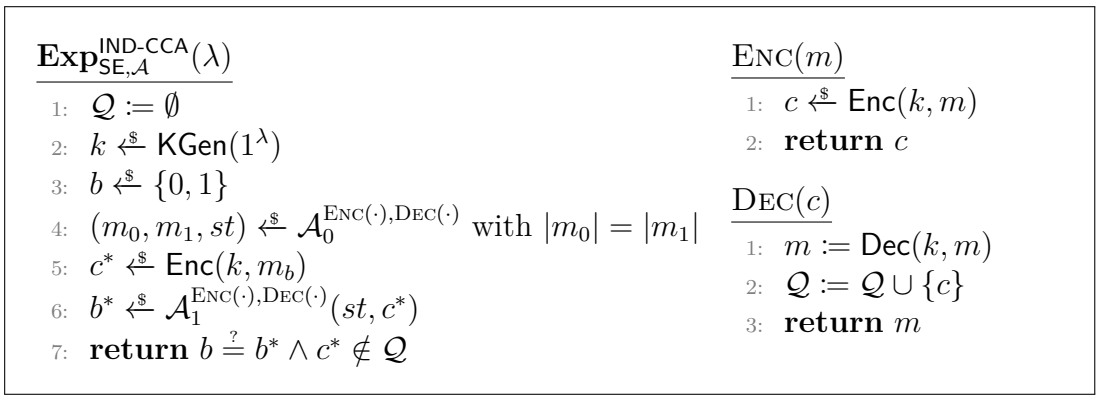

Figure 2.2: Security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ for IND-CCA-security of symmetric encryption.

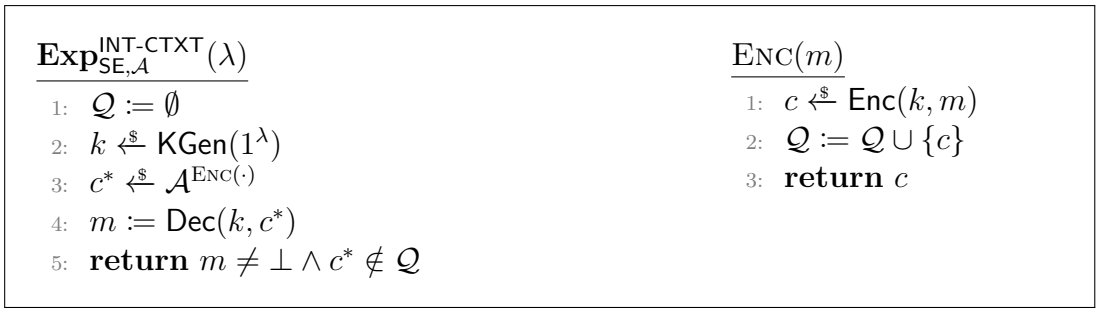

Figure 2.3: Security experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ for INT-CTXT-security of symmetric encryption.

We say SE is IND-CCA-secure if $\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

After defining security notions for the *secrecy* of symmetric encryption, we now turn to *integrity*, which aims to ensure that an adversary cannot tamper with a ciphertext without the receiver of the message noticing the tampering. For this we consider the security notion of *ciphertext integrity*, for which we give the formal security experiment in Figure 2.3.

Definition 6. The advantage of an adversary $\mathcal{A}$ against the *ciphertext integrity* (INT-CTXT) of an SE scheme $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SE},\mathcal{A}}(\lambda) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SE},\mathcal{A}}(\lambda) = 1].$$

We say SE is INT-CTXT-secure if $\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

An SE scheme that is IND-CCA-secure and INT-CTXT-secure is often also called *authenticated encryption* (AE). We can further extend AE to *authenticated*

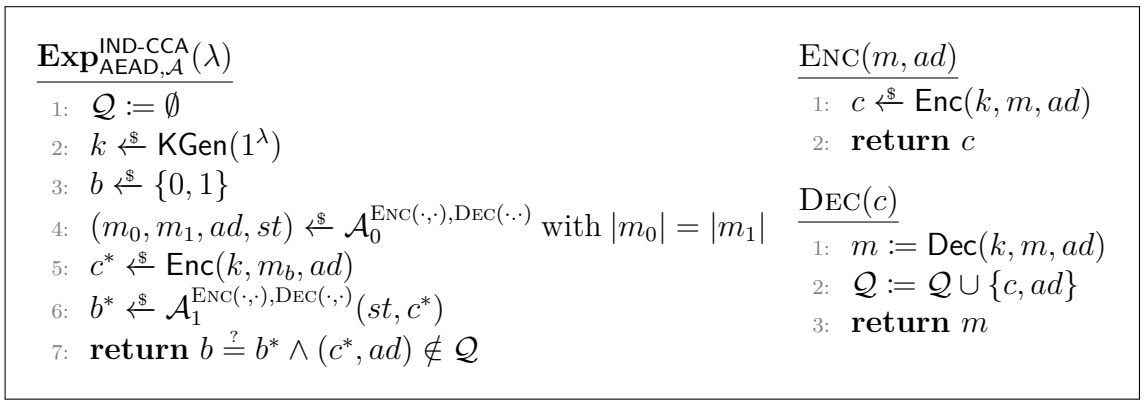

Figure 2.4: Security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ for IND-CCA-security of authenticated encryption with associated data.

encryption with associated data (AEAD), where the encryption algorithm takes an additional input ad for which integrity but no secrecy will be ensured by the ciphertext.

Definition 7. An *authenticated encryption with associated data scheme* consists of three PPT algorithms $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and associated data space $\mathcal{AD}$.

$\mathsf{KGen}(1^\lambda)$**:** takes as input a security parameter λ and outputs a key $k \in \mathcal{K}$.

$\mathsf{Enc}(k, m, ad)$**:** takes as input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and associated data $ad \in \mathcal{AD}$, and outputs a ciphertext $c \in \mathcal{C}$.

$\mathsf{Dec}(k, c, ad)$**:** takes as input a key $k \in \mathcal{K}$, a ciphertext $c \in \mathcal{C}$, and associated data $ad \in \mathcal{AD}$, and outputs a message $m \in \mathcal{M}$ or an error symbol $\perp$.

We say a scheme AEAD is *correct* if for all $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, all $m \in \mathcal{M}$, and all $ad \in \mathcal{AD}$ it holds that

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m, ad), ad) = m.$$

Again, we consider IND-CCA- and INT-CTXT-security (see Figure 2.4 and 2.5).

Definition 8. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *indistinguishability of ciphertexts under chosen ciphertext attacks* (IND-CCA) of an AEAD scheme $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \coloneqq \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) = 1\right] - 1 \right|.$$

We say AEAD is IND-CCA-secure if $\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

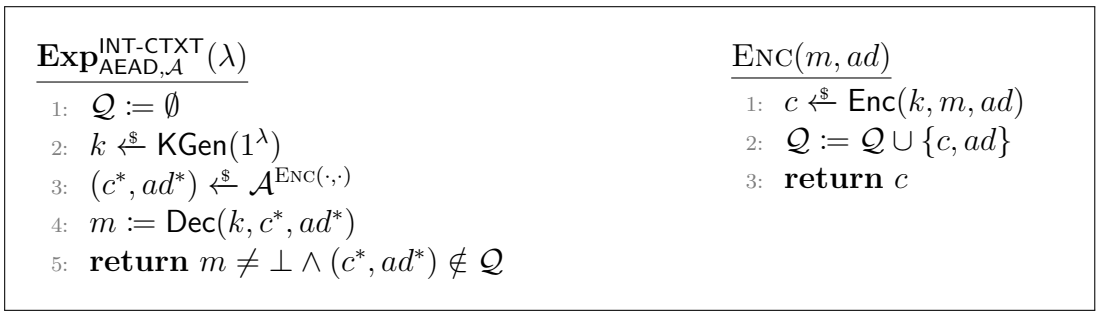

Figure 2.5: Security experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ for INT-CTXT-security of authenticated encryption with associated data.

Definition 9. The advantage of an adversary $\mathcal{A}$ against the *ciphertext integrity* (INT-CTXT) of an AEAD scheme $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) = 1].$$

We say AEAD is INT-CTXT-secure if $\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

2.3.3 Asymmetric Encryption

An *asymmetric* or *public-key encryption* (PKE) scheme allows parties to communicate without having to first establish a shared key. Instead a party generates a key pair consisting of a public and a secret key. The public key is published and used by other parties to encrypt messages. The corresponding secret key is then used to decrypt ciphertexts.

Definition 10. A *public-key encryption scheme* consists of three PPT algorithms $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K} = \mathcal{PK} \times \mathcal{SK}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$.

$\mathsf{KGen}(1^\lambda)$**:** takes as input a security parameter λ and outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$.

$\mathsf{Enc}(\mathsf{pk}, m)$**:** takes as input a public key $\mathsf{pk} \in \mathcal{PK}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.

$\mathsf{Dec}(\mathsf{sk}, c)$**:** takes as input a secret key $\mathsf{sk} \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$ or an error symbol $\bot$.

We say a scheme PKE is *correct* if for all $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ and all $m \in \mathcal{M}$ it holds that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m.$$

$\underline{\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)}$

1: $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2: $b \xleftarrow{\$} \{0,1\}$
3: $(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}_0^{\textsc{Dec}(\cdot)}(\mathsf{pk})$ with $|m_0| = |m_1|$
4: $c^* \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m_b)$
5: $b^* \xleftarrow{\$} \mathcal{A}_1^{\textsc{Dec}(\cdot)}(st, c^*)$
6: **return** $b \stackrel{?}{=} b^*$

Figure 2.6: Security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ for IND-CPA-security of public-key encryption.

$\underline{\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)}$

1: $\mathcal{Q} = \emptyset$
2: $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
3: $b \xleftarrow{\$} \{0,1\}$
4: $(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}_0^{\textsc{Dec}(\cdot)}(\mathsf{pk})$ with $|m_0| = |m_1|$
5: $c^* \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m_b)$
6: $b^* \xleftarrow{\$} \mathcal{A}_1^{\textsc{Dec}(\cdot)}(st, c^*)$
7: **return** $b \stackrel{?}{=} b^* \wedge c^* \notin \mathcal{Q}$

$\underline{\textsc{Dec}(c)}$

1: $m \coloneqq \mathsf{Dec}(k, c)$
2: $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{c\}$
3: **return** m

Figure 2.7: Security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ for IND-CCA-security of public-key encryption.

Similarly to symmetric encryption, we again consider IND-CPA and IND-CCA security. A difference in the security experiments for PKE is that there is no encryption oracle. This is due to the fact that the adversary receives the public key and can hence encrypt messages by itself without the help of an oracle. The formal security experiments are given in Figures 2.6 and 2.7.

Definition 11. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *indistinguishability of ciphertexts under chosen plaintext attacks* (IND-CPA) of a PKE scheme $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda) \coloneqq \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1\right] - 1 \right|.$$

We say PKE is IND-CPA-secure if $\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

Definition 12. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *indistinguishability of ciphertexts under chosen ciphertext attacks* (IND-CCA) of a PKE

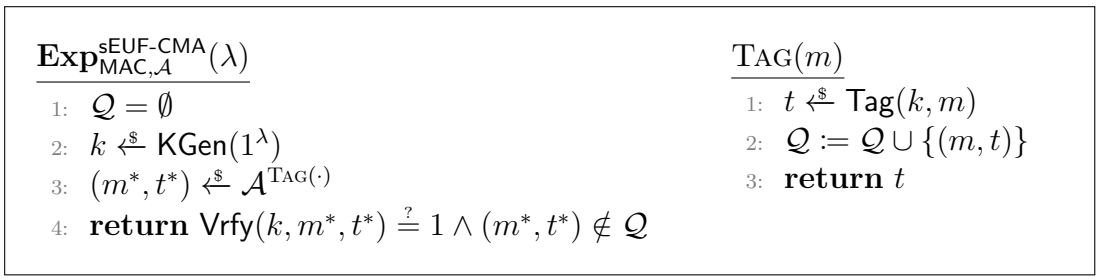

Figure 2.8: Security experiment $\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$ for sEUF-CMA-security of message authentication codes.

scheme $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{A}}(\lambda) := \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1\right] - 1 \right|.$$

We say PKE is IND-CCA-secure if $\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

2.3.4 Message Authentication Codes

A *message authentication code* (MAC) allows parties to ensure the authentication of their communication. For this, the parties rely on a shared key, that can be used to compute *tags* over messages and to verify tags.

Definition 13. A *message authentication code* consists of three PPT algorithms $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vrfy})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, and tag space $\mathcal{T}$.

$\mathsf{KGen}(1^\lambda)$**:** takes as input a security parameter λ and outputs a key $k \in \mathcal{K}$.

$\mathsf{Tag}(k, m)$**:** takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a tag $t \in \mathcal{T}$.

$\mathsf{Vrfy}(k, m, t)$**:** takes as input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$ and outputs 0 or 1.

We say a scheme MAC is *correct* if for all $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ and all $m \in \mathcal{M}$ it holds that

$$\mathsf{Vrfy}(k, m, \mathsf{Tag}(k, m)) = 1.$$

Informally speaking, we require for security that it should be difficult to forge a valid tag for any message. We formalize this via the standard notion of *strong existential unforgeability under chosen message attacks* (sEUF-CMA), for which we depict the security experiment in Figure 2.8.

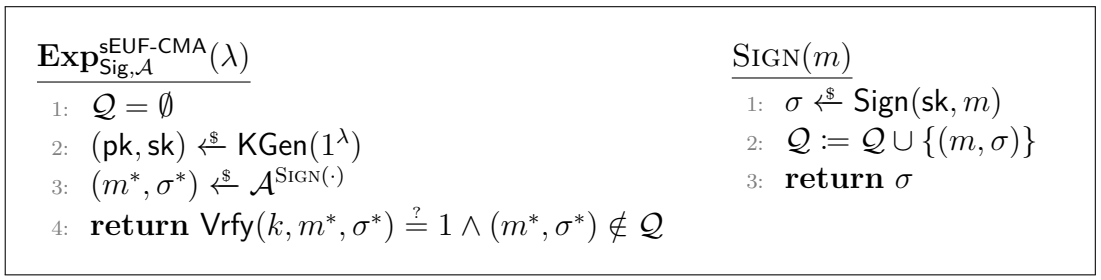

Figure 2.9: Security experiment $\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(\lambda)$ for sEUF-CMA-security of digital signatures.

Definition 14. The advantage of an adversary $\mathcal{A}$ against the *strong existential unforgeability under chosen message attacks* (sEUF-CMA) of a MAC scheme $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vrfy})$ is defined as

$$\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(\lambda) = 1\right].$$

We say MAC is sEUF-CMA-secure if $\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

2.3.5 Digital Signatures

Digital signatures are the asymmetric analogue to MACs. The goal of achieving authentication is the same for both primitives, however in contrast to MACs, digital signatures do not rely on a shared key but on a key pair consisting of a public and a secret key. The secret key is used to sign messages and anyone in possession of the corresponding public key can verify the signature.

Definition 15. A *digital signature* consists of three PPT algorithms $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ with key space $\mathcal{K} = \mathcal{PK} \times \mathcal{SK}$, message space $\mathcal{M}$, and signature space Σ.

$\mathsf{KGen}(1^\lambda)$**:** takes as input a security parameter λ and outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** takes as input a secret key $\mathsf{sk} \in \mathcal{SK}$ and a message $m \in \mathcal{M}$ and outputs a signature $\sigma \in \Sigma$.

$\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$**:** takes as input a public key $\mathsf{pk} \in \mathcal{PK}$, a message $m \in \mathcal{M}$, and a signature $\sigma \in \Sigma$ and outputs 0 or 1.

We say a scheme Sig is *correct* if for all $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ and all $m \in \mathcal{M}$ it holds that

$$\mathsf{Vrfy}(k, m, \mathsf{Sign}(k, m)) = 1.$$

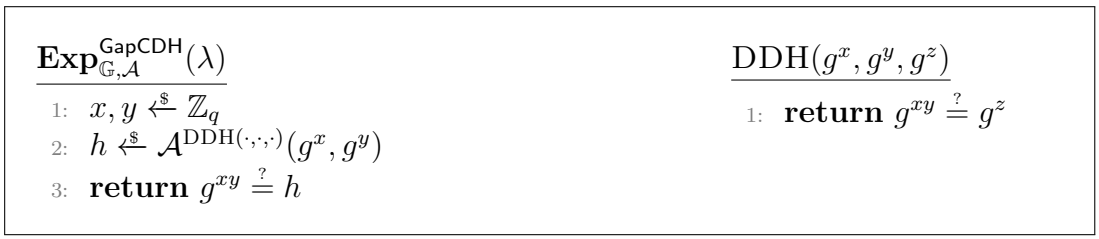

Figure 2.10: Security experiment $\mathbf{Exp}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{A}}(\lambda)$ for the GapCDH problem relative to a group $\mathbb{G} = \langle g \rangle$ of prime order q.

Security-wise we again require that it should be difficult to forge a valid signature, which is formalized in the security experiment in Figure 2.9.

Definition 16. The advantage of an adversary $\mathcal{A}$ against the *strong existential unforgeability under chosen message attacks* (sEUF-CMA) of a digital signature $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ is defined as

$$\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(\lambda) = 1\right].$$

We say Sig is sEUF-CMA-secure if $\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

2.4 Computational Problems

In this section we recap some well-known computational problems that we use for our security proofs. All problems we introduce here are variants of the computational *Diffie–Hellman* (DH) problem, which stems from the seminal work by Diffie and Hellman [DH76].

Gap Diffie–Hellman. We begin with the GapCDH problem [GJK21]. Throughout this section, let $\mathbb{G}$ be a cyclic group of prime order q with generator g. The *computational Diffie–Hellman* (CDH) problem is the task of computing g^{xy} given two random elements g^x and g^y. In the GapCDH problem the adversary is additionally given access to an oracle $\mathrm{DDH}(\cdot,\cdot,\cdot)$ that takes as input three group elements g^x, g^y, and g^z and outputs 1 if $z = xy$ and 0 otherwise (see Figure 2.10).

Definition 17. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The advantage of an adversary $\mathcal{A}$ in the *gap computational Diffie–Hellman* (GapCDH) problem is defined as

$$\mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathbf{Exp}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{A}}(\lambda) = 1\right].$$

We say the GapCDH problem is hard relative to $\mathbb{G}$ if $\mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

$\underline{\mathbf{Exp}^{\mathsf{StDH}}_{\mathbb{G},\mathcal{A}}(\lambda)}$

1: $x, y \xleftarrow{\$} \mathbb{Z}_q$
2: $h \xleftarrow{\$} \mathcal{A}^{\mathrm{DDH}(\cdot,\cdot)}(g^x, g^y)$
3: **return** $h \stackrel{?}{=} g^{xy}$

$\underline{\mathrm{DDH}(g^y, g^z)}$

1: **return** $g^{xy} \stackrel{?}{=} g^z$

Figure 2.11: Security experiment $\mathbf{Exp}^{\mathsf{StDH}}_{\mathbb{G},\mathcal{A}}(\lambda)$ for the StDH problem relative to a group $\mathbb{G} = \langle g \rangle$ of prime order q.

$\underline{\mathbf{Exp}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda)}$

1: $x, y_1, \ldots, y_n \xleftarrow{\$} \mathbb{Z}_q$
2: $q \coloneqq 0$
3: $S \xleftarrow{\$} \mathcal{A}^{\mathrm{DDH}(\cdot,\cdot,\cdot,\cdot),\mathrm{CDH}(\cdot)}(g^x, g^{y_1}, \ldots, g^{y_n})$
4: **return** $S \stackrel{?}{=} \{(g^{y_i}, g^{xy_i}) \mid i \in \{1, \ldots, n\}\}$
$\wedge\, |S| = q + 1 \wedge q < n$

$\underline{\mathrm{DDH}(h, h^x, h^y, h^z)}$

1: **return** $h^{xy} \stackrel{?}{=} h^z$

$\underline{\mathrm{CDH}(h)}$

1: $q \coloneqq q + 1$
2: **return** h^x

Figure 2.12: Security experiment $\mathbf{Exp}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda)$ for the OMDH problem relative to a group $\mathbb{G} = \langle g \rangle$ of prime order q.

Strong Diffie–Hellman. Next, we introduce the *strong Diffie-Hellman* problem [ABR01]. This is essentially the same as the GapCDH problem, except that the first input to the DDH oracle is fixed to the challenge value g^x. The experiment is depicted in Figure 2.11.

Definition 18. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The advantage of an adversary $\mathcal{A}$ in the *strong Diffie–Hellman* (StDH) problem is defined as

$$\mathbf{Adv}^{\mathsf{StDH}}_{\mathbb{G},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathbf{Exp}^{\mathsf{StDH}}_{\mathbb{G},\mathcal{A}}(\lambda) = 1\right].$$

We say the StDH problem is hard relative to $\mathbb{G}$ if $\mathbf{Adv}^{\mathsf{StDH}}_{\mathbb{G},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

One-more Diffie–Hellman. Finally, we consider the *one-more Diffie–Hellman* (OMDH) problem [JKKX16], which is another variant of the CDH problem. Here, the adversary gets n values $g^{y_1}, \ldots, g^{y_n}$ as input and can use the $\mathrm{CDH}(\cdot)$ oracle to get the value g^{xy_i} for all but one y_i. Additionally, it is given access to an even more powerful oracle $\mathrm{DDH}(\cdot,\cdot,\cdot,\cdot)$ than in the GapCDH problem, where it can choose the base to which the discrete logarithm of the three group elements is taken (see Figure 2.12).

Definition 19. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The advantage of an adversary $\mathcal{A}$ in the *one-more Diffie–Hellman* (OMDH) problem is defined as

$$\mathbf{Adv}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda) = 1\right].$$

We say the n-OMDH problem is hard relative to $\mathbb{G}$ if $\mathbf{Adv}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

2.5 The Random Oracle Model

The *random oracle model* (ROM) [BR93] is a tool that is commonly used in the security proofs of many cryptographic schemes using hash functions. The ROM is often used when the scheme cannot be proven secure under standard assumptions such as collision resistance. In the ROM, a hash function is idealized as a truly random function with consistent input/output behavior to which all parties and the adversary have oracle access. In security proofs the random oracle is typically implemented as a large input/output table that is filled via *lazy-sampling*, which means that the table is initially empty and whenever the random oracle receives an input that it has not seen previously, it samples a new random output and adds the pair of input and output to the table. A very useful feature of the ROM, which is enabled by the lazy-sampling, is the *programmability*. Programmability denotes that a reduction simulating a random oracle to an adversary does not have to sample truly random outputs but may also choose the outputs for some queries to its liking as long as they are indistinguishable from a uniformly random output. This is a very important proof technique and we will also make use of it in this thesis.

Notably, a security proof in the ROM does not translate to practice if the given hash function does not closely approximate the random oracle, meaning that it does not behave almost like a random function. A notable example are hash functions based on the Merkle–Damgård construction [Dam90] such as MD5 [Riv92], SHA1 [EJ01], and SHA2 [EH11], which all suffer from length extension attacks [Tsu92]. Nevertheless, a proof in the ROM is still better than no proof at all and moreover implies that in order to break the scheme in practice, an adversary must exploit weaknesses in the hash function.[3] For this reason, the ROM is still widely accepted as a useful paradigm.

[3] Assuming no other idealizing assumption is made in the proof.

Part I

Key Confirmation

3 On the Concrete Security of Key Confirmation

Author's contribution. This chapter is based on currently unpublished joint work with Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager, which is an extension of the results by Gellert, Gjøsteen, Jacobsen, and Jager [GGJJ23]. The author of this thesis contributed the impossibility result in Section 3.6, which extends the impossibility result of [GGJJ23] to a larger class of protocols than in [GGJJ23]. The idea for the result evolved from discussions with Kristian Gjøsteen and Tibor Jager, and the author of this thesis contributed the formal write up of the result.

3.1 Introduction

From Weak to Full Forward Security. To upgrade an AKE protocol from weak forward security to full forward security, there are typically two main approaches for a generic transform: either using the key confirmation paradigm we already introduced in Section 1.1 or authenticating the protocol messages with digital signatures. The key confirmation approach is very efficient, and adds only a negligible communication and computational cost to the underlying protocol as the key confirmation values can be MACs, or extra output from the PRF that generated the session key. The digital signature approach on the other hand is simple and generic, but does not yield the most efficient protocols, since the signatures add communication and computational overhead to the underlying protocol.

From Implicit to Explicit Authentication. The notions of weak and full forward security are also closely connected to *implicit* and *explicit* authentication in key exchange protocols [BPR00, BG11, DFW20]. Implicit authentication ensures to a protocol participant that only its intended communication peer will be able to derive the same session key. However, it does not guarantee that this peer actually derived the session key or in fact participated in the protocol at all. By contrast, explicit authentication ensures that the intended communication partner is indeed online and has actively participated in the protocol.

Besides being able to turn weakly secure protocols into fully forward secure protocols, another feature of key confirmation (and digital signatures) is that it can also generically convert an implicitly authenticated protocol into an explicitly

authenticated one.[1]

Efficient protocols and digital signatures. In this chapter we aim to construct highly efficient key exchange protocols with full forward security. In doing so, we will not focus on protocols based on digital signatures. There are two reasons for this.

The first is that tightly secure key exchange protocols whose user long-term keys are digital signature keys require signature schemes with tight security in a *multi-user setting with adaptive corruptions* [BHJ+15]. This is because standard single-user security notions, such as textbook EUF-CMA security, would incur an inherent linear security loss in the number of users. While there exist several constructions of suitable schemes [BHJ+15, Bad14, GJ18, DGJL21, PW22, HLWG23], all known constructions are much less efficient than standard signature schemes.

The second reason is the transition to post-quantum cryptography. We currently do not have candidates of post-quantum digital signature schemes that can serve as generic one-to-one replacements for classical digital signature scheme in many applications. This is because current post-quantum schemes have significantly more overhead, either in terms of computation or communication. Among those recently selected for standardization by NIST, SPHINCS+ [HBD+22] has very large signatures and is considered too slow for many applications.[2] Other schemes have better computational performance [LDK+22, PFH+22], but suffer from relatively large public keys and signatures. Therefore, while digital signatures are a standard building block of real-world key exchange protocols, such as TLS 1.3 [Res18], there is currently a trend to entirely avoid digital signatures in future key exchange protocols. For example, the KEMTLS protocol [SSW20] is a proposal for a quantum-secure TLS-replacement that uses a key encapsulation mechanism (KEM) instead of a signature scheme. KEMs can currently be constructed much more efficiently in the post-quantum setting. Other examples are the generic protocol designs based on KEMs by Pan *et al.* [PWZ23] and Hövelmanns *et al.* [HKSU20], which both provide only *weak* forward security. While a signature scheme provides explicit authentication in a straightforward way, a KEM alone provides only *implicit* authentication via the communication partner's ability to correctly decrypt an encapsulated key. Explicit authentication and full forward security can then be achieved by adding key confirmation messages. Hence, even though key confirmation has been used for more than two decades, understanding its concrete security is becoming particularly relevant in the post-quantum setting.

[1] Implicitly authenticated protocols often have weak forward security, and explicitly authenticated protocols often have full forward security. However, we note that these are distinct notions. That is, there are protocols having explicit authentication but no forward security [BR94, Fig. 2], and fully forward secure protocols without explicit authentication, e.g. [BG11, Protocol 4] and [CF15, Fig. 3].

[2] See `https://blog.cloudflare.com/post-quantum-future/`, for example.

The exact security of key confirmation. Suppose Π is an arbitrary key exchange protocol providing weak forward security and implicit authentication. The protocol participants exchange a session key using Π and from this derive key confirmation messages as well as a new session key using a pseudorandom function (PRF). They then exchange and verify the confirmation messages before outputting the new session key. Call this extended protocol Π^+. Intuitively, protocol Π^+ should achieve full forward security and explicit authentication via a *tight* reduction to the weak forward security and implicit authentication of protocol Π as well as the multi-user security of the PRF. Indeed, this is the claim of Theorem 6 in [CCG+19]. Unfortunately, this claim turns out to be wrong. In fact, as we will show, for certain natural protocols, such as the protocol from Cohn-Gordon et al. [CCG+19] and HMQV [Kra05], adding key confirmation messages like this must *necessarily* lose a factor of U, where U is the number parties in the protocol.

To explain the flaw in [CCG+19], we first describe the high-level idea of their security reduction from protocol Π^+ to protocol Π. The reduction uses Test or Reveal queries to get the session keys from Π and uses these to simulate the key confirmation messages of protocol Π^+. However, the reduction must decide which session keys it will obtain via Reveal queries and which keys it will obtain via Test queries. The standard strategy would be to guess which session the adversary (against Π^+) will test. But this approach cannot be used in [CCG+19] as it would immediately incur a linear security loss in the number of users *times* the number of sessions per user. Instead, the reduction proceeds as follows: once a session has reached an accepting state in the underlying AKE protocol Π, the reduction will base its decision on which query to use on the *current* freshness of the session. If the session is not fresh, it will issue a Reveal query. If the session is fresh, it will issue a Test query.

The problem with this strategy is that the freshness notion is with respect to protocol Π, which is only guaranteeing *weak* forward security. Recall that in a weak forward security model the adversary is forbidden from both actively modifying the messages in a Test session *and* revealing the long-term secret of its peer (refer to Section 3.3 for formal definitions). This is due to an unavoidable attack first described by [BPR00] (see [BG11] and [CF15] for further discussions). However, in the context of the reduction of [CCG+19], this becomes an issue since the adversary against Π^+ is *not* restricted in this way, while the reduction is.

To illustrate the issue (and the attack), consider an implicitly authenticated Diffie–Hellman based protocol where Alice and Bob exchange DH shares g^x and g^y, then derive their session key from g^{xy} and some combination of their long-term keys with g^x and g^y. In the attack, the adversary $\mathcal{A}$ first impersonates Alice towards Bob by creating the DH share g^x on her behalf. Once Bob receives this message he creates its own DH share g^y and accepts in protocol Π. Since $\mathcal{A}$ has not (yet) revealed the long-term key of Alice, Bob is at this point still fresh in protocol Π according to the weak forward security model. Consequently, the reduction will issue a Test query in order to simulate its key confirmation message in protocol Π^+. However, if $\mathcal{A}$ now reveals the long-term key of Alice, then Bob will no longer

be fresh (in protocol Π). At this point the reduction is stuck. This means that the reduction in Theorem 6 of [CCG+19] does not work.

Our contributions. While the reduction of [CCG+19] does not work, can the result nevertheless be salvaged? Unfortunately, no. The notions of weak forward secrecy and implicit authentication turns out to be too weak to be *tightly* upgraded to full forward secrecy and explicit authentication by simply adding key confirmation messages. Specifically, we show that a tight reduction from full forward secrecy and explicit authentication to weak forward secrecy and implicit authentication is impossible for a large class of compilers and protocols of interest for practical applications. In particular, this includes the common key confirmation message compiler discussed above and the key exchange protocol of [CCG+19]. We prove this using a meta-reduction described in more detail below.

On the other hand, by considering what the actual *end goal* of [CCG+19] was—to create as efficient as possible key exchange protocols having full forward security and explicit authentication, with optimal tightness—we can in fact recover their intended result by a rearranging of arguments. Here, tightness is with respect to the lowest-level building block of the protocol. In the case of [CCG+19] this is the strong Diffie–Hellman (StDH) assumption (see Section 2.4). It was shown in [CCG+19] that a large class of DH-based implicitly authenticated key exchange protocols must lose a factor of U when reducing to StDH, where U is the number of parties. If the reduction from Π^+ to Π had been tight, as mistakenly claimed in [CCG+19], then the overall result would have been a protocol Π^+ with full forward security and an optimal tightness loss of U to the StDH assumption. However, in light of our impossibility result, the best one can hope for using this approach is a loss of U^2, since there is a tightness loss of U going from Π^+ to Π and a tightness loss of U going from Π to StDH.

But this begs the question: if we know from the beginning that we have to lose a factor of at least U, is there some other way of structuring our arguments in order to avoid a quadratic loss? Interestingly, the solution is to first reduce the security of protocol Π^+ to an even *weaker* notion of implicit security for protocol Π, taking the "hit" of U here. This weaker notion is a *selective security* variant of the key exchange game where the adversary must commit to a single party it will not reveal the long-term key of.[3] Establishing that full forward secrecy and explicit authentication can be reduced generically to this selective security variant with only a loss of U is our second main contribution.

Finally, if the selective security notion can be satisfied *tightly* by protocol Π from some underlying hardness assumption, then, by using the generic result above, we obtain a protocol Π^+ with full forward security and explicit authentication losing

[3]This is related to the selective security notion from [KPW13], but the two notions are technically incomparable (see Remark 1). Moreover, in [KPW13] the adversary commits to *both* parties *and their sessions* involved in the event. This incurs a quadratic security loss, making it unsuitable for our purposes.

only and overall factor of U. That is, we provide a *modular* approach towards creating protocols with full forward security (and explicit authentication): first prove that the initial protocol Π satisfies selective security, then add key confirmation messages to upgrade to full forward security. If the first can be obtained tightly, then our generic result shows that the latter is obtained with a linear tightness loss. Our third main contribution is to show that the selective security notion can indeed be attained tightly. We illustrate this with the [CCG+19] protocol and show that it satisfies selective security tightly from the strong Diffie–Hellman assumption. Together with the generic upgrade theorem, this re-establishes their originally claimed result, namely that full forward security can be obtained from the StDH assumption with only a linear loss.

In summary, our main results are:

- (Section 3.4) We prove that a key exchange protocol can be upgraded from *selective* weak forward security and implicit authentication, to full forward security and explicit authentication using key confirmation. The proof has a linear security loss in the number of parties U, which is optimal by the impossibility result in Section 3.6.

  One consequence of the generic upgrade theorem is that future key exchange protocols can be designed towards the goal of selective security. As illustrated by the examples in Section 3.5, this may simplify proofs significantly.

- (Section 3.5) We show that the Diffie–Hellman-based protocol from [CCG+19] tightly satisfies the selective security notion, illustrating that the overall proof strategy achieves our goal of a protocol with full forward security and a linear security loss.

- (Section 3.6) Finally, we give an impossibility result stating to the effect that all security proofs showing that key confirmation upgrades weak forward security to full forward security, must have a security loss of at least U.

Basic idea of the impossibility result. Our impossibility result shows essentially that if one constructs a protocol Π^+ from an underlying protocol Π by extending it with two additional key confirmation messages, and if the security analysis of Π^+ includes a reduction $\mathcal{R}$ to the weak forward security and implicit authentication of Π, then $\mathcal{R}$ loses a factor which is at least linear in the number of parties U.

We prove this result with a meta-reduction $\mathcal{M}$ that runs the reduction $\mathcal{R}$ as a subroutine by simulating a high-advantage adversary $\mathcal{A}$ for $\mathcal{R}$. The main challenge of the meta-reduction is to properly simulate $\mathcal{A}$, as this should actually be impossible to do efficiently if the protocol is secure. Hence, we have to define the adversary such that on the one hand it breaks security of the compiled protocol, and thus is a valid adversary, and on the other hand $\mathcal{M}$ is able to simulate $\mathcal{A}$ *efficiently* in almost all cases. A bound on the probability that $\mathcal{M}$ is *not* able to simulate $\mathcal{A}$ then can be used to derive a bound on the security loss of the

reduction. This is the standard meta-reduction technique, as used for example in [Cor02, PV05, HJK12, FF13, LW14, BJLS16, FJS19, GGJJ23].

The main technical novelty in this thesis is how we implement this technique. Before explaining this, we first have to sketch the hypothetical adversary $\mathcal{A}$ that our meta-reduction will simulate (see Section 3.6.2 for a precise description of $\mathcal{A}$). It proceeds as follows.

1. $\mathcal{A}$ receives U public keys from its Π^+ experiment and chooses a public key at random to be distinguished.

2. $\mathcal{A}$ asks its experiment to start one session for each public key, such that the sessions have distinct partner keys and their partners are supposed to send the first key confirmation message.

3. $\mathcal{A}$ simulates partners for these sessions by creating protocol messages that it sends to the experiment, until it receives key confirmation messages.

   The main purpose of the first three steps is to force the reduction to output (properly distributed) key confirmation messages for all U sessions.

4. $\mathcal{A}$ corrupts the secret keys of all but the distinguished public key and verifies that they are correct, aborting if not.

   The main purpose of this step is to force $\mathcal{R}$ to "know" all secret keys corresponding to all public keys except for the distinguished key.

5. $\mathcal{A}$ checks that it received correct key confirmation messages from all the sessions in Step 3 and aborts if not.

   The Π^+ security experiment will always send correct key confirmation messages and secret keys, but a reduction might not be able to do this. This is where we will obtain the inherent tightness loss of the reduction.

6. Finally, $\mathcal{A}$ (somehow) computes the session key and the second key confirmation message for the session with the distinguished public key as its partner sends it, which then accepts. $\mathcal{A}$ then tests it and compared the returned key to the real session key, winning the game.

Note that the adversary performs potentially inefficient computations in Steps 3, 4, 5 and 6. However, following the standard meta-reduction technique, it is sufficient to describe such a hypothetical adversary, because a black-box reduction $\mathcal{R}$ should still be able to leverage $\mathcal{A}$ to solve some computationally hard problem, such as breaking the security of Π.

Now let us consider our approach to simulate this hypothetical adversary towards a reduction $\mathcal{R}$ in our meta-reduction $\mathcal{M}$. The first main difficulty arises in Step 3 when sending messages without having corrupted the corresponding key. This is because the protocol messages of Π^+ produced by $\mathcal{A}$ may not be efficiently computable without knowing the sending user's long-term secret key.

Hence, $\mathcal{M}$ cannot efficiently compute them. To overcome this issue we need new meta-reduction techniques.

Our first new technique uses that we have defined the hypothetical adversary such that all messages it needs to produce (until its last step) are messages of the underlying protocol Π. This is the case, because our definition of $\mathcal{A}$ guarantees that $\mathcal{R}$ outputs the *first* key confirmation message for all sessions, and all previous messages are messages of Π. Then we use that the meta-reduction runs $\mathcal{R}$ as a subroutine, and that $\mathcal{R}$ itself interacts with the security experiment of Π. Our new idea is to leverage the security experiment of Π, by letting $\mathcal{M}$ create "hidden sessions" in this experiment. These sessions are invisible to $\mathcal{R}$, in the sense that $\mathcal{R}$ is not aware that they exist. On the one hand, this provides us with a leverage to efficiently provide $\mathcal{M}$ with the messages that it cannot necessarily efficiently compute on its own. On the other hand, we have to proceed carefully, because we have to make sure that the hidden sessions do not interfere with the interaction between $\mathcal{R}$ and the security experiment of Π.

The second difficulty lies in Step 4. For the long-term secret keys, we will argue that the reduction must have forwarded all corruption queries to its own security experiment for Π, such that all these users are corrupted there, too. If the reduction did not for some query, then either it did not provide a correct secret key and we can thus efficiently simulate $\mathcal{A}$, or it outputs a long-term secret key which is correct with some non-negligible probability, and the corresponding user is uncorrupted in the security experiment of Π. In the latter case, we show how $\mathcal{R}$ could use this secret key to break the security of Π directly.

The third main difficulty lies in Step 5, in the verification of the key confirmation messages. Again, it is not straightforward how this step can be efficiently implemented by $\mathcal{M}$, because even though $\mathcal{A}$ (and thus $\mathcal{M}$) knows all secret keys (except the one corresponding to the distinguished public key) from the corrupt queries in Step 4, it knows neither the secret key corresponding to the distinguished public key, because this key was not corrupted, nor the randomness that the experiment (or the reduction) has used in order to create the message received when $\mathcal{A}$ has impersonated that user. Our new approach here is to let $\mathcal{M}$ inspect the queries made by $\mathcal{R}$ to simulate the key confirmation messages, and apply a careful case distinction.

- If there exists any session for which $\mathcal{R}$ outputs a key confirmation message, but $\mathcal{R}$ did not reveal the session key, then $\mathcal{M}$ assumes that the corresponding key confirmation message is incorrect and thus is able to properly simulate $\mathcal{A}$ by letting it abort. To argue that this is a proper simulation, we show that if the key confirmation is correct, but the corresponding session not revealed we can use it to break Π.

- We also need to consider the possibility that the adversary causes our "hidden sessions" to become partnered, which could be problematic since we may have revealed the session keys of these hidden sessions. This is a technical problem which we reduce to match security for the underlying protocol.

Once we are sure neither of these events happen, the only way the meta-reduction does not correctly abort is if the reduction correctly predicted our distinguished session, which happens with probability at most $1/U$.

Evading the impossibility result. We emphasize that our impossibility result only rules out reductions from the "standard" full forward security of Π^+ to the "standard" weak forward security of Π. While this matches how key confirmation is commonly used to achieve full forward security (cf. [Kra05, Yan13a, GGJJ23]), one possible way to circumvent it could be to either aim for a weaker, yet possibly equally useful notion of full forward security. Alternatively, one could start from a notion of weak forward security that provides stronger properties, but is possibly equally efficiently achievable. This was indeed done in a recent work by Pan *et al.* [PRZ24]. They show how to construct *tightly* secure protocols via key confirmation, thus seemingly contradicting our impossibility results. Their main idea is to circumvent the impossibility result by starting from a weakly-secure protocol that provides a new, non-standard security notion called *one-way verifiable weak forward secrecy* (OW-VwFS). Essentially, the *verifiability* means that an adversary is able to efficiently verify that a given session key belongs to a given session. In contrast, our impossibility results consider underlying protocols with *standard* indistinguishability-based weak forward secrecy (wFS). As also explained in [PRZ24], the *verifiability* of their new OW-VwFS notion is stronger than standard wFS, and thus makes it possible to bypass the impossibility.

Both results together nicely explain the possibility and impossibility of achieving tight security with the key confirmation paradigm: if the underlying weakly-secure protocol additionally provides verifiability in the sense of OW-VwFS, then one can obtain a tightly and fully forward-secure protocol with the approach of [PRZ24]. However, in general this is not the case, and thus our impossibility results show that the standard key confirmation paradigm is inherently lossy.

Another approach to evade the impossibility result could be to exploit that we require that the underlying protocol Π has unique and efficiently verifiable secret keys. This condition seems inessential, but there are concrete constructions where protocol messages may, without compromising security, depend on the secret key such that it is easy to see if two sessions use the same secret key. For example, a reduction might embed some trivial redundancy to secret keys, such as a random string which is sent along with every message. This might allow the reduction to distinguish whether an adversary uses the same secret key as the reduction. While we do not see how this could allow a reduction to avoid the impossibility result, it would allow a reduction to detect our meta-reduction techniques, forming a technical obstruction to our proof. However, one can easily generalize our result from unique to *rerandomizable* keys, using techniques from [HJK12, BJLS16]. Essentially, the rerandomization ensures that a meta-reduction can turn a given secret key for public key pk into a uniformly random secret key among all valid secret keys for pk. We omit this, since this is a relatively straightforward generalization.

3.2 Definitions

The formalism and definitions we use to model key exchange protocols are adapted from de Saint Guilhelm et al. [DFW20]. Unlike the traditional Bellare–Rogaway [BR94, BR95, BPR00] and (e)CK models [CK01, LLM07], security in this model is not formulated as a single all-in-one game that implicitly captures all the properties a protocol should have. Instead, security is split into many smaller definitions that each captures a single "atomic" security property. This leads to a slight increase in the number of definitions, as well as the number of proofs one have to carry out in order to establish a protocol as "secure". On the other hand, the advantage of this approach is that each definition/property is much simpler and focused, and the corresponding proofs similarly simple.

3.2.1 Syntax

A *key exchange protocol* is a tuple of stateless algorithms $(\mathsf{KGen}, \mathsf{Init}, \mathsf{Run})$ where KGen is the long-term key generation algorithm; Init creates a *session state* at party i having intended peer j and role role, and returns this session's initial message (empty if a responder role); and Run takes as input a session state st and a message m and outputs an updated state st' and response message m'.

Session state. A *session state* st consists of the following variables.

- $\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}, \bot\}$ – indicates the status of the key exchange run; initialized to $\bot$ and indicates a running, non-completed, session.
- $\mathsf{key} \in \{0,1\}^* \cup \{\bot\}$ – the local session key derived during the key exchange run; set once $\mathsf{accept} = \mathsf{true}$.
- $\mathsf{role} \in \{\mathsf{init}, \mathsf{resp}\}$ – the role of the session in the key exchange run.
- party – the party identity to which this session belongs.
- peer – the party identity of the intended peer for this key exchange run.
- sk – the secret long-term key of the party this session belongs to.
- pk – the public long-term key of the intended peer of the session.
- $\mathsf{transcript}$ – the (ordered) transcript of all messages sent and received by session s. We use $\mathsf{transcript}^-$ to denote the transcript minus the last message.
- aux – auxiliary protocol specific state, such as internal randomness and ephemeral values.

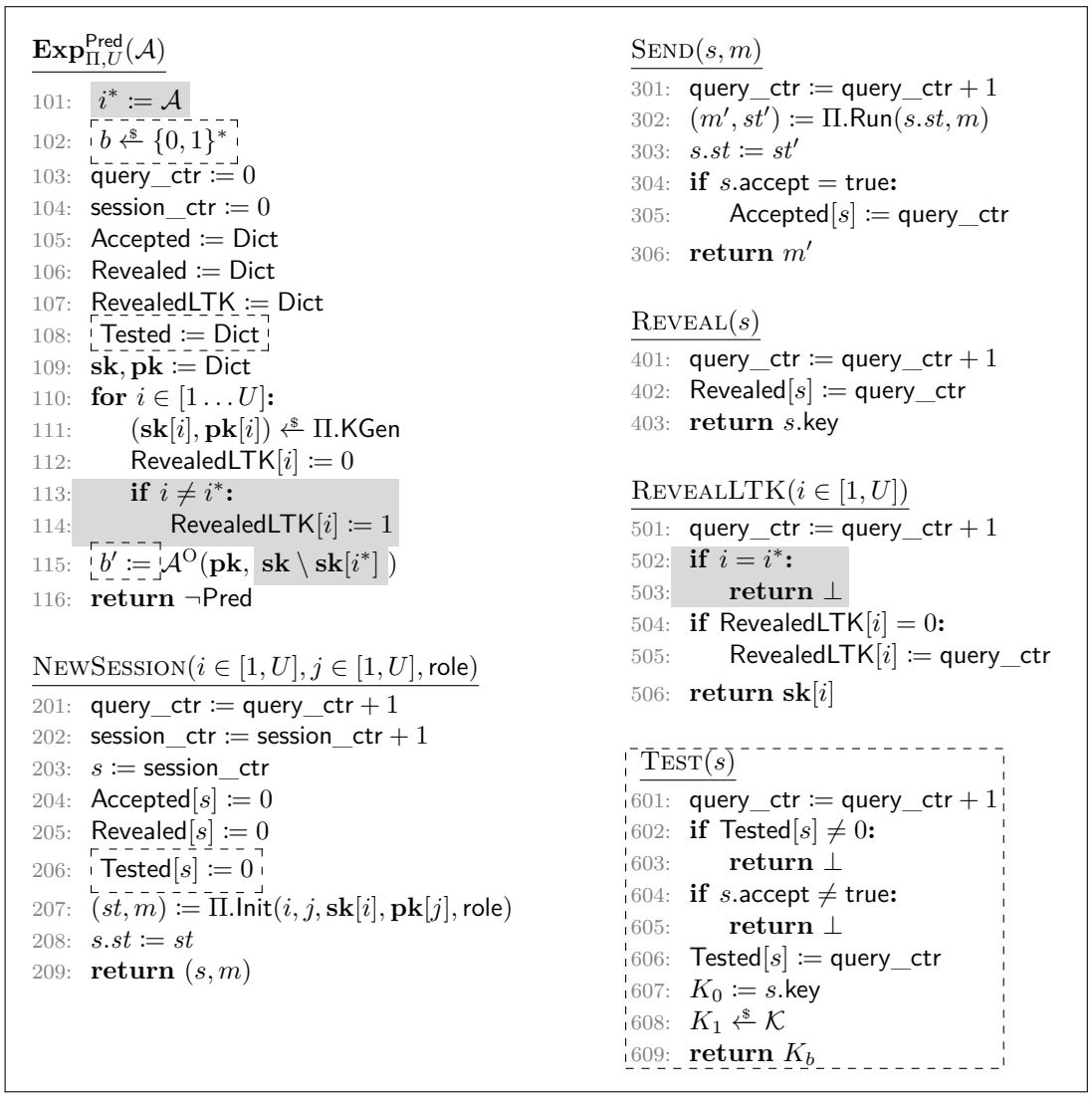

Figure 3.1: Generic experiment parameterized on predicate Pred, where $\mathcal{A}$ can make the queries in $\mathrm{O} = \{\textsc{NewSession}, \textsc{Send}, \textsc{Reveal}, \textsc{RevealLTK}, \textsc{Test}\}$. Code in dashed boxes is only for the key secrecy game; code in filled boxes is only for the selective key secrecy game. The notation $s.st \coloneqq st'$ means to assign all the variables in st' to the corresponding variables associated with session s. Dict defines an associative array.

Security experiment We shall use the generic formal experiment $\mathbf{Exp}^{\mathsf{Pred}}_{\Pi,U}(\mathcal{A})$ given in Fig. 3.1 to define the various security properties of a key exchange protocol (see Section 3.3). The experiment is parameterized on a *security predicate* Pred that captures the security property being modeled. The experiment uses a number of counters, variables and collections for bookkeeping purposes.

- query_ctr – incremented for each query made by the adversary. Used to

order events in time; needed to define (full) forward secrecy.

- session_ctr – incremented for each new session created. Each session state is associated with a unique session number which functions as an administrative label for that session (state). The session number is also given to the adversary which can use it as an opaque handle to refer to a given session in its queries. We use the notation "$s.\mathsf{x}$" to refer to the variable x of the session state identified by the administrative session number s. Note that the adversary cannot "dereference" a session number in order to obtain internal variables of the session state.

- Accepted, Tested, Revealed, RevealedLTK – associative arrays that record when a session accepted, was tested, or when its session or long-term key was revealed.

Common predicates. It will be useful to introduce a number of predicates on the security experiment.

Definition 20 (Origin sessions)**.** A (possibly non-accepted) session s' is an *origin-session* for an accepted session s if predicate $\mathsf{Orig}(s, s')$ holds true, where

$$\mathsf{Orig}(s, s') \iff s'.\mathsf{transcript} \in \{s.\mathsf{transcript}, s.\mathsf{transcript}^{-}\}. \tag{3.1}$$

Definition 21 (Partnering)**.** Two sessions s, s' are *partners* if they have matching conversations; that is, if the predicate $\mathsf{Partner}(s, s')$ holds true, where

$$\mathsf{Partner}(s, s') \iff s.\mathsf{transcript} = s'.\mathsf{transcript}. \tag{3.2}$$

Like [DFW20] we do not require partners to agree upon each other's identities. This is an authentication property which will be covered by other definitions in Section 3.3. Unlike [DFW20] we use matching conversations instead of abstract session identifiers as our partnering mechanism. This is mainly done for the sake of concreteness and is not a fundamental difference, although certain well-known pitfalls need to be avoided when using matching conversations [LS17].

Definition 22 (SameKey)**.** The predicate $\mathsf{SameKey}(s, s')$ holds true if the sessions both have established a session key and they are equal, that is

$$\mathsf{SameKey}(s, s') \iff [s.\mathsf{key} = s'.\mathsf{key} \neq \perp]. \tag{3.3}$$

Definition 23 (Authentication fresh)**.** A session is *authentication fresh* if the long-term key of its intended peer has not been revealed, that is:

$$\mathsf{aFresh}(s) \iff \mathsf{RevealedLTK}[s.\mathsf{peer}] = 0. \tag{3.4}$$

Finally, we define freshness predicates used for the key secrecy games. These come in two flavors: *weak forward secrecy* and *full forward secrecy* [BPR00]. Common to both is that the adversary cannot reveal the session key of a tested session or its partner. The difference is how long-term key leakage is handled. For weak forward secrecy the adversary is forbidden from revealing the long-term key of a session's peer if it was actively interfering in the protocol run of the session (indicated by the lack of an origin-session for the session in question). For full forward secrecy this restriction is lifted, provided the leak happened *after* the session in question accepted.

Definition 24 (Session key freshness)**.** Let $s.\mathsf{peer} = j$. The $\mathsf{kFreshWFS}(s)$ (resp. $\mathsf{kFreshFFS}(s)$) predicate hold if:

$$\mathsf{Revealed}[s] = 0 \tag{3.5}$$

$$\forall s' :: \mathsf{Partner}(s, s') \implies \mathsf{Revealed}[s'] = 0 \wedge \mathsf{Tested}[s'] = 0 \tag{3.6}$$

$$\text{(wFS)} \quad \{s' \mid \mathsf{Orig}(s, s')\} = \emptyset \implies \mathsf{aFresh}(s) \tag{3.7}$$

$$\text{(fFS)} \quad \{s' \mid \mathsf{Orig}(s, s')\} = \emptyset \implies \mathsf{aFresh}(s) \vee (\mathsf{RevealedLTK}[j] > \mathsf{Accepted}[s]) \tag{3.8}$$

3.3 Protocol security properties

This section defines the security properties a secure key exchange protocol ought to have. The breakdown follows that of [DFW20] and consists of: soundness properties (match and key-match soundness); various authentication properties (implicit/explicit key and entity authentication); and session key secrecy. An application will typically require all of these properties. Refer to [DFW20] for further discussion and background.

Recall that the security experiment in Fig. 3.1 is parameterized on a predicate Pred. In the following we define concrete predicates for each of the security properties above. For each predicate there is also an associated advantage notion.

3.3.1 Match soundness

Match soundness is primarily a sanity check on the choice of partnering mechanism. Namely, partnered sessions should derive the same session key (3.9); and sessions will at most have one partner (3.10).

Definition 25 (Match soundness)**.** The Match predicate evaluates to 1 iff $\forall s, s', s''$:

$$\mathsf{Partner}(s, s') \implies \mathsf{SameKey}(s, s') \tag{3.9}$$

$$(\mathsf{Partner}(s, s') \wedge \mathsf{Partner}(s, s'')) \implies s' = s'' \tag{3.10}$$

The *match soundness advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) = 1] \tag{3.11}$$

3.3.2 Key-match soundness

Key-match soundness (KMSound) is basically the converse of Match soundness. While Match soundness says that partners should have equal session keys, KMSound says that sessions having equal session keys should be partners.

Definition 26 (Key-match soundness)**.** The KMSound predicate evaluates to 1 if and only if

$$\forall s :: (\mathsf{aFresh}(s) \wedge s.\mathsf{accept}) \implies \forall s' :: (\mathsf{SameKey}(s, s') \implies \mathsf{Partner}(s, s')) \quad (3.12)$$

The *key-match soundness advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) = 1]. \quad (3.13)$$

3.3.3 Implicit key authentication

Implicit key authentication stipulates that two sessions that derive the same session key should agree upon *whom* they are sharing this key with.

Definition 27 (Implicit key authentication)**.** The iKeyAuth predicate evaluates to 1 if and only if

$$\forall s :: s.\mathsf{accept} \implies \forall s' :: (\mathsf{SameKey}(s, s') \implies s.\mathsf{peer} = s'.\mathsf{party})$$

The *implicit key authentication advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) = 1]. \quad (3.14)$$

3.3.4 Explicit key authentication

Explicit key authentication stipulates that any two sessions that derive the same session key should agree upon whom they are sharing this key with (as for implicit key authentication), and as long as the session is authentication fresh some other session deriving the same session key should exist.

Obviously, the session that sends the last message can never guarantee that this message arrives at its destination, which means that this session can only achieve the notion of *almost-full* key authentication, namely that an origin session should exist and any origin session that has derived a session key has derived the same key. A session that receives the last message, however, can guarantee that another session exists that has derived the same key, and thereby achieve *full* key authentication.

Let $\mathcal{L}_{\mathsf{rcv}}$ denote the collection of all sessions that *receive* the last message of the protocol, and let $\mathcal{L}_{\mathsf{send}}$ denote the collection of all sessions that *send* the last message of the protocol.

Definition 28 (Explicit key authentication)**.** The fexKeyAuth predicate (resp. afexKeyAuth predicate) evaluates to 1 if and only if

$$\forall s \in \mathcal{L}_{\mathsf{rcv}}\,(\text{resp. } \mathcal{L}_{\mathsf{send}}) :: s.\mathsf{accept} \implies \forall s' :: (\mathsf{SameKey}(s,s') \Rightarrow s.\mathsf{peer} = s'.\mathsf{party})$$

$$\wedge$$

$$\text{(full)} \qquad \mathsf{aFresh}(s) \Rightarrow \exists s' :: \mathsf{SameKey}(s,s')$$

$$\text{(almost-full)} \qquad \mathsf{aFresh}(s) \Rightarrow \exists s' :: \Big(\mathsf{Orig}(s,s') \wedge [s'.\mathsf{key} \neq \perp \implies \mathsf{SameKey}(s,s')]\Big)$$

The *full (resp. almost-full) explicit key authentication advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A}) = 1] \tag{3.15}$$

$$\mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A}) = 1] \tag{3.16}$$

3.3.5 Explicit entity authentication

Explicit *entity* authentication is almost identical to explicit *key* authentication, the only difference being that the former is based on the Partner predicate while the latter is based on the SameKey predicate. Basically, explicit key authentication says that if a session with an honest peer accepts then there *is* some other session holding the same session key, while explicit entity authentication says that if a session with an honest peer accepts then it *has* a partner session.

Explicit key authentication and explicit entity authentication are closely related, as shown in [DFW20].

Definition 29 (Explicit entity authentication)**.** The fexEntAuth predicate (resp. afexEntAuth predicate) evaluates to 1 if and only if

$$\forall s \in \mathcal{L}_{\mathsf{rcv}}\,(\text{resp. } \mathcal{L}_{\mathsf{send}}) :: s.\mathsf{accept} \implies \forall s' ::(\mathsf{Partner}(s,s') \implies s.\mathsf{peer} = s'.\mathsf{party})$$

$$\wedge$$

$$\text{(full)} \qquad \mathsf{aFresh}(s) \implies \exists s' :: \mathsf{Partner}(s,s')$$

$$\text{(almost-full)} \quad \mathsf{aFresh}(s) \implies \exists s' :: \Big(\mathsf{Orig}(s,s') \wedge [s'.\mathsf{accept} \implies \mathsf{Partner}(s,s')]\Big)$$

The *full (resp. almost-full) explicit entity authentication advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) = 1] \tag{3.17}$$

$$\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) \coloneqq \Pr[\mathbf{Exp}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) = 1] \tag{3.18}$$

3.3.6 Key secrecy

Key secrecy is defined as usual with the adversary using a TEST query to get the real session key or a random key of a session. The adversary may make multiple test queries, and they all share the same challenge bit, so that either all TEST queries return real keys, or all TEST queries return random (and independently)

sampled keys. Our experiment does not prevent the adversary from making TEST queries for sessions that are not key fresh, so we need to account for this in the definition of advantage (called the *penalty-style* in [RZ18]).

Definition 30 (Key secrecy)**.** If $\forall s \in \mathsf{Tested} :: \mathsf{kFreshWFS}(s) = \mathsf{true}$ (resp. $\mathsf{kFreshFFS}(s) = \mathsf{true}$), the KeySecWFS (resp. KeySecFFS) predicate returns 1 if and only if $b' = b$. Else it returns b. The *weak (resp. full) forward key secrecy advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{A}) \coloneqq \left|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{A}) = 1] - 1\right| \tag{3.19}$$

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi,U}(\mathcal{A}) \coloneqq \left|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi,U}(\mathcal{A}) = 1] - 1\right| \tag{3.20}$$

Selective key secrecy. The *selective* key secrecy experiment is defined over the experiment given in Fig. 3.1, where now the code inside the blue boxes is included. In the selective security experiment the adversary has to commit to one party it will not reveal the long-term key of throughout the game.

Definition 31 (Selective key secrecy)**.** If $\forall s \in \mathsf{Tested} :: \mathsf{kFreshWFS}(s) = \mathsf{true}$, the SelKeySecWFS predicate returns 1 if and only if $b' = b$. Else it returns b. The *selective key secrecy advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{A}) \coloneqq \left|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{A}) = 1] - 1\right|. \tag{3.21}$$

Remark 1. Contrary to what one might expect, ordinary key secrecy does *not* trivially reduce to selective key secrecy via a standard guessing argument (with a tightness loss of U). In particular, an adversary that starts by revealing *all* long-term keys will make a reduction to selective key secrecy unable to simulate the one key it committed to. This makes our selective security security notion incomparable to the selective notion of [KPW13], where the selected long-term key only has to be involved in some event, not necessarily stay unrevealed throughout.

3.4 The security of adding key confirmation

Let Π denote an arbitrary key exchange protocol, and let Π^+ denote the protocol that extends Π by adding key confirmation messages from each side as illustrated in Fig. 3.2. Conventionally, the key confirmation messages are derived from the session key of Π using a PRF (and possibly a MAC) but in order to simplify the later analysis we assume that Π produces session keys of the form (k, t, t') directly. Protocol Π^+ is then derived from Π simply by defining its session key to be k, and the key confirmation tags to be t and t'. Using this trick we can relate the security of protocol Π^+ purely to the security of protocol Π without having to rely on PRFs or MACs.

Unfortunately, defining Π^+ in terms of the key triple output by Π introduces one technicality. We will often want to make an assertion of the form "if s and s' have

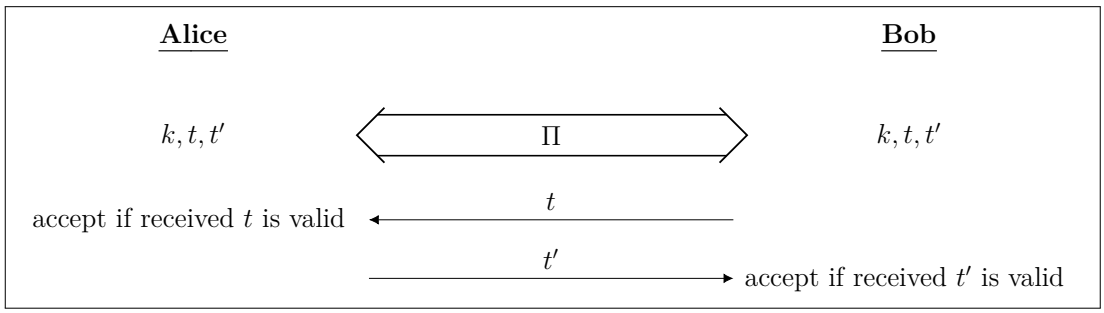

Figure 3.2: Protocol Π^+ obtained by extending protocol Π with key confirmation tags. All session variables in Π^+ are inherited from Π, except for accept which is defined as shown. The session sending the last message in protocol Π sends tag t, and the session receiving the last message in protocol Π sends tag t'.

equal keys in protocol Π^+ (meaning k), then they also have equal keys in protocol Π (meaning (k, t, t'))". While this assertion easily follows in practice[4], in the generality we have presented Π and Π^+ above the assertion does not automatically follow. To cleanly state and prove our generic results we therefore introduce the implication "equal $k \implies$ equal (k, t, t')" as an explicit security property.

To this end, let prefix : $\{0,1\}^* \to \{0,1\}^*$ be a function that returns a prefix of a particular length (left unspecified) from its argument and define

$$\mathsf{SamePrefix}(s, s') \iff [s.\mathsf{key}, s'.\mathsf{key} \neq \perp \wedge \mathrm{prefix}(s.\mathsf{key}) = \mathrm{prefix}(s'.\mathsf{key})]. \quad (3.22)$$

Definition 32 (Same prefix security)**.** The PreEqAllEq predicate evaluates to 1 if and only if

$$\forall s, s' : \mathsf{SamePrefix}(s, s') \implies \mathsf{SameKey}(s, s'). \quad (3.23)$$

The *same prefix advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1]. \quad (3.24)$$

Remark 2. As mentioned above, proving same prefix security for a concrete protocol will typically be straightforward assuming the keys are derived using a reasonable function. However, it is also possible to avoid the notion altogether (even for completely generic protocols) using the proof technique of Lemma 1 in Section 3.4.2. Specifically, if s and s' have equal keys in protocol Π^+ but not in protocol Π, then this allows to break the (selective) key secrecy of protocol Π, albeit with a tightness loss in the number of parties U.

[4]For example if (k, t, t') is derived from the session transcript using a function for which getting a collision just in k is unlikely, such as an extendable-output function or a random oracle

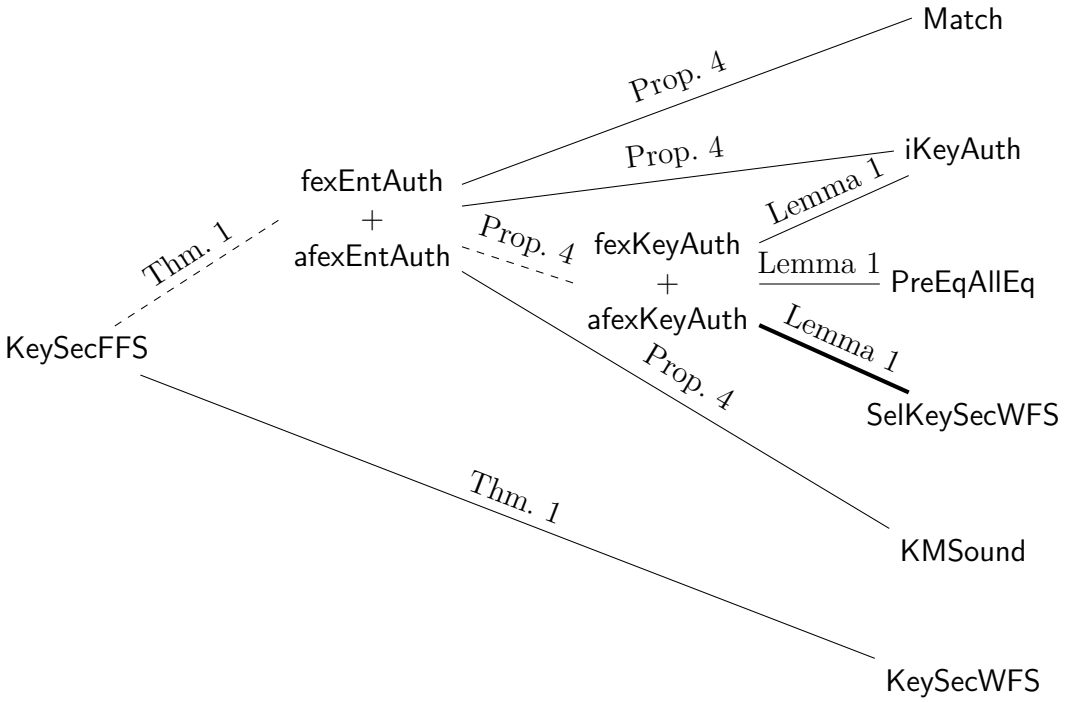

Figure 3.3: Steps in the proof of Theorem 1. A dashed line from X to Y means that Y is not part of the statement of X, but only an intermediate step inside its proof. A bold line from X to Y means that the reduction loses a factor of U. Note that some reductions include additional intermediate steps not shown.

3.4.1 Main result

We now state the first main theorem of this chapter: a protocol with weak forward security can be upgraded to full forward security by adding key confirmation messages with a linear security loss in the number of parties U. By the impossibility result in Section 3.6, the linear loss is optimal.

Theorem 1. *Let $\mathcal{A}$ be an adversary against key secrecy for Π^+. Then there exist adversaries $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_6$, all with about the same runtime as $\mathcal{A}$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) \leq 4 \cdot U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_1) + 8 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \frac{4US}{2^{\mathsf{taglen}}} \\ + 4 \cdot \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_3) + 4 \cdot \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_4) \\ + 12 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_5) + 4 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6),\end{aligned}$$

where taglen *is the length of the key confirmation tags used by Π^+ and S is the number of sessions.*

The outline of the proof of Theorem 1 is shown in Fig. 3.3. At a high level the proof consists of two parts: one where all accepting sessions with peers whose

long-term keys are unrevealed have an origin session, and one where they don't. In the first case full forward key secrecy of protocol Π^+ reduces straightforwardly to the weak forward key secrecy of protocol Π. The main challenge is to deal with the second case, namely to prove that protocol Π^+ achieves explicit entity authentication. In fact, the main technical tool for this is to prove that Π^+ achieves explicit *key* authentication, which is where we use the *selective* key secrecy notion. The proof of explicit key authentication is the focus of Section 3.4.2.

Proof (of Theorem 1). We prove that

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + 2 \cdot \mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + 4 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6) \tag{3.25}$$

from which the result will follow by Proposition 4.

Let Win_{G_i} denote that $\mathcal{A}$ wins in Game i, i.e., $b' = b$ and all TEST sessions are key fresh.

Game $\mathbf{G}_0$: Original game. This is the original key secrecy game for protocol Π^+.

Game $\mathbf{G}_1$: Reject sessions without origin. In this game all sessions s having honest peers (i.e., $\mathsf{aFresh}(s) = \mathsf{true}$), but not having an origin session reject all tags. In particular, this means that these sessions will never accept in protocol Π^+.

Claim 1.

$$\Pr[\mathsf{Win}_{G_0}] \leq \Pr[\mathsf{Win}_{G_1}] + \mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \tag{3.26}$$

Proof. Let E be the event that a session with an honest peer accepts in protocol Π^+, but without having an origin session. Since Game $\mathbf{G}_0$ and Game $\mathbf{G}_1$ are identical unless event E occurs, it is sufficient to bound $\Pr[E]$.

Suppose s is a session that triggers event E. In particular, (i) $s.\mathsf{accept} = \mathsf{true}$, (ii) $\mathsf{aFresh}(s) = \mathsf{true}$, and (iii) $\{s' \mid \mathsf{Orig}(s', s)\} = \emptyset$. By the last property, s also doesn't have a partner, and by the first two properties this means that either $\mathsf{fexEntAuth}$ is violated (in case $s \in \mathcal{L}_{\mathsf{rcv}}$) or $\mathsf{afexEntAuth}$ is violated (in case $s \in \mathcal{L}_{\mathsf{send}}$). □

Game $\mathbf{G}_2$: Replace keys in accepting sessions. In this game all $\mathsf{kFreshWFS}$ sessions *which accept* have their session keys replaced by random.

Claim 2.

$$\Pr[\mathsf{Win}_{G_1}] \leq \Pr[\mathsf{Win}_{G_2}] + 2 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6). \tag{3.27}$$

Proof. Algorithm $\mathcal{B}_6$ begins by drawing a random bit b_{sim}, then simulates the following game for $\mathcal{A}$. Whenever $\mathcal{A}$ makes a query pertaining to protocol Π, then $\mathcal{B}_6$ forwards it to its own (weak forward secrecy) game. When a session s accepts in protocol Π, then $\mathcal{B}_6$ simulates protocol Π^+ for $\mathcal{A}$ as follows.

- If s is not kFreshWFS in protocol Π then $\mathcal{B}_6$ issues a $\textsc{Reveal}(s)$ query to its key secrecy game and uses the returned key (k, t, t') to simulate s's tags in protocol Π^+.
- If s is kFreshWFS, but does not have an origin session (in Π), then $\mathcal{B}_6$ issues a $\textsc{Reveal}(s)$ query to its key secrecy game and uses the returned key (k, t, t') to simulate the tag *sent* by s. If s *receives* a tag, then it is simply rejected.
- If s is kFreshWFS and has an origin session s' (in Π), then,
  - if s' haven't accepted in Π yet (and thus haven't been issued a $\textsc{Test}$ or $\textsc{Reveal}$ query), then $\mathcal{B}_6$ issues a $\textsc{Test}$ query to s to obtain a key (k, t, t');
  - if, on the other hand, s' have already accepted, then by the previous cases $\mathcal{B}_6$ must already have issued a $\textsc{Test}$ or $\textsc{Reveal}$ query to s', which returned a key (k, t, t').

  In either case, $\mathcal{B}_6$ uses the returned key (k, t, t') to simulate both how s sends *and* receives tags.

To answer $\mathcal{A}$'s $\textsc{Reveal}$ queries (in Π^+), $\mathcal{B}_6$ uses the "k" element of the tuples it obtained above. To answer $\mathcal{A}$'s $\textsc{RevealLTK}$ queries, $\mathcal{B}_6$ simply forward these to its own game. To answer $\mathcal{A}$'s $\textsc{Test}$ queries, $\mathcal{B}_6$ answers as follows.

- If the session has already been tested, or has not accepted yet, return $\perp$.
- If $b_{sim} = 0$ then $\mathcal{B}_6$ returns key from the (k, t, t') tuple it previously obtained for this session, as described above.
- If $b_{sim} = 1$ then $\mathcal{B}_6$ returns a random key $\widetilde{k}$.

Finally, when $\mathcal{A}$ stops and outputs a bit b', then $\mathcal{B}_6$ outputs 1 to its own key secrecy game if and only if $b' = b_{sim}$.[5]

We first claim that if the secret bit in $\mathcal{B}_6$'s key secrecy game is 0 (hence $\mathcal{B}_6$'s $\textsc{Test}$ queries are answered with real session keys), then $\mathcal{B}_6$ perfectly simulates either Game $\mathbf{G}_1$ or Game $\mathbf{G}_2$, depending on the bit b_{sim}.

Note first that if s is not kFreshWFS, then $\mathcal{B}_6$ obtains s's actual key in protocol Π, hence simulates protocol Π^+ correctly. Second, if s is kFreshWFS, but does not have an origin session, then s rejects any tag, and thus never accepts in protocol Π^+. This is exactly what happens after the change in Game $\mathbf{G}_1$. Finally, if s is kFreshWFS and has an origin session, then it behaves exactly as in Game $\mathbf{G}_1$ if $b_{sim} = 0$ (since then $\mathcal{B}_6$ is using the actual

[5] Assuming all tested sessions are key fresh (according to kFreshFFS). Otherwise $\mathcal{B}_6$ simply outputs a random bit.

key from Π) and exactly as in Game $\mathbf{G}_2$ if $b_{sim} = 1$ (since then $\mathcal{B}_6$ is using a completely random key).

On the other hand, if the secret bit in $\mathcal{B}_6$'s key secrecy game is 1, then $\mathcal{B}_6$ simulates Game $\mathbf{G}_2$ independently of what b_{sim} is. Specifically, this means that b_{sim} is completely hidden from $\mathcal{A}$ in this case.

Consequently, assuming the secret bit in $\mathcal{B}_6$'s key secrecy game is b, we have

$$\begin{aligned}\Pr[\mathcal{B}_6 \text{ wins}] &= \Pr[\mathcal{B}_6 \text{ wins} \mid b = 0 \wedge b_{sim} = 0] \cdot \frac{1}{4} \\ &\quad + \Pr[\mathcal{B}_6 \text{ wins} \mid b = 0 \wedge b_{sim} = 1] \cdot \frac{1}{4} \qquad (3.28) \\ &\quad + \Pr[\mathcal{B}_6 \text{ wins} \mid b = 1] \cdot \frac{1}{2} \\ &= \Pr[\mathsf{Win}_{\mathbf{G}_1}] \cdot \frac{1}{4} + (1 - \Pr[\mathsf{Win}_{\mathbf{G}_2}]) \cdot \frac{1}{4} + \frac{1}{4} \qquad (3.29) \\ &= \Pr[\mathsf{Win}_{\mathbf{G}_1}] \cdot \frac{1}{4} - \Pr[\mathsf{Win}_{\mathbf{G}_2}] \cdot \frac{1}{4} + \frac{1}{2}. \qquad (3.30)\end{aligned}$$

Hence,

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6) = |2 \cdot \Pr[\mathcal{B}_6 \text{ wins}] - 1| = \frac{1}{2} \cdot |\Pr[\mathsf{Win}_{\mathbf{G}_1}] - \Pr[\mathsf{Win}_{\mathbf{G}_2}]| \qquad (3.31)$$

which proves the claim. □

Concluding the proof of Theorem 1. By the change in Game $\mathbf{G}_2$ all session keys of $\mathsf{kFreshWFS}$ sessions are now random, hence $\Pr[\mathsf{Win}_{\mathbf{G}_2}] = 1/2$. Combining the bounds from (3.26) and (3.27), and multiplying by 2, yields (3.25), from which Theorem 1 follows by Proposition 4. □

3.4.2 Implicit to explicit key authentication

In this section, we establish that explicit key authentication can be based on *selective* key secrecy, implicit key authentication, and same prefix security. This is a key technical result needed to restore the tight security of the explicitly authenticated protocol of [CCG+19]. The use of selective security may also have further applications in constructing highly efficient explicitly authenticated key exchange protocols with full forward secrecy in the future.

Lemma 1. *Let $\mathcal{A}$ be an adversary against full (resp. almost full) explicit key authentication for Π^+. Then there exists an adversary $\mathcal{B}_2$ against selective key secrecy and an adversary $\mathcal{B}_1$ against implicit key authentication and same prefix security, both with the same runtime as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_1) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_1) + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_2) + \frac{US}{2^{\mathsf{taglen}}},$$

$$\mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_1) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_1) + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_2) + \frac{US}{2^{\mathsf{taglen}}},$$

where taglen *is the length of the key confirmation tags used by* Π^+ *and* S *is the number of sessions.*

Since the proofs of full and almost full key authentication are virtually identical, we only prove the fist bound. We need to deal with two cases. The first case considers attacks on explicit authentication that result from breaking implicit authentication of the underlying protocol Π. This case does not incur a tightness loss.

The second case considers attacks on explicit authentication that rely on breaking the weak forward secrecy of the underlying protocol Π. The important point is that in order to break explicit authentication, the partner long-term key must be unrevealed at the point in time where authentication is broken. This means that the session will be fresh at the time authentication is broken, which means that we can deduce the challenge bit at the point in time where authentication is broken. Any subsequent reveal of the partner long-term key can therefore be ignored.

Proof. The proof is structured as a sequence of games. Let Win_{G_i} denote the event that $\mathcal{A}$ wins in Game i. Winning in this case means that full explicit key authentication in (3.15) from Definition 28 does not hold.

Game $\mathbf{G}_0$: Original game. This is the original game for protocol Π^+. We have that

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) = \Pr[\mathsf{Win}_{\mathbf{G}_0}]. \tag{3.32}$$

Game $\mathbf{G}_1$: Reject non-authenticated sessions. We modify the game so that if (3.15) does not hold for the Π part of a session of Π^+, then that session never accepts. Let $\mathsf{Except}_{\mathbf{G}_1}$ be the event that this happens.

It is immediate that until $\mathsf{Except}_{\mathbf{G}_1}$ happens, Game $\mathbf{G}_1$ proceeds exactly as Game $\mathbf{G}_0$, so

$$|\Pr[\mathsf{Win}_{\mathbf{G}_1}] - \Pr[\mathsf{Win}_{\mathbf{G}_0}]| \leq \Pr[\mathsf{Except}_{\mathbf{G}_1}]. \tag{3.33}$$

We create an adversary $\mathcal{B}_1$ against implicit key authentication for Π that runs a copy of $\mathcal{A}$ and uses its experiment to run the Π part of Π^+. When a session of Π outputs a session key, $\mathcal{B}_1$ reveals the session key and uses that to simulate sending and receiving the key confirmation messages. Let $\mathsf{Win}_{\mathcal{B}_1}$ denote the probability that $\mathcal{B}_1$ wins.

It is immediate that $\mathcal{B}_1$ and its experiment together simulate the experiment in Game 0 perfectly with respect to the copy of $\mathcal{A}$ run by $\mathcal{B}_1$. Since SameKey for Π^+ implies SamePrefix for Π, if (3.15) does not hold for Π^+ in an execution, either it will not hold when we consider the game as an execution of

Π, or PreEqAllEq will not hold when we consider the game as an execution of Π. In other words,

$$\Pr[\mathsf{Except}_{\mathbf{G}_1}] \leq \Pr[\mathsf{Win}_{\mathcal{B}_1}^{\mathsf{iKeyAuth}}] + \Pr[\mathsf{Win}_{\mathcal{B}_1}^{\mathsf{PreEqAllEq}}]. \tag{3.34}$$

Game $\mathbf{G}_2$: Guess public key. We modify the game by sampling $j \in \{1, 2, \ldots, U\}$ at the start. Let $\mathsf{Win}'_{\mathbf{G}_2}$ be the event that $\mathsf{Win}_{\mathbf{G}_2}$ happens and one session for which authentication is broken has the jth key as its peer's public key. Clearly,

$$\Pr[\mathsf{Win}'_{\mathbf{G}_2}] \geq \frac{1}{U} \Pr[\mathsf{Win}_{\mathbf{G}_2}] = \frac{1}{U} \Pr[\mathsf{Win}_{\mathbf{G}_1}]. \tag{3.35}$$

Game $\mathbf{G}_3$: Use random tags. We modify the game so that if (3.15) holds for a session of Π^+ that has the jth key as its peer public key but it has no origin session, then that session samples random tags to use for the Π^+ part of the protocol, instead of the tags output by Π.

It is immediate that

$$\Pr[\mathsf{Win}'_{\mathbf{G}_3}] \leq \frac{S}{2^{\mathsf{taglen}}}. \tag{3.36}$$

We create an adversary $\mathcal{B}_2$ against selective key secrecy for Π that runs a copy of $\mathcal{A}$ and uses its experiment to run the Π part of Π^+, simulating the sending and receiving of key confirmation messages as modified in Game $\mathbf{G}_2$, further modified as follows:

- At the start, $\mathcal{B}_2$ selects an integer $j \in \{1, 2, \ldots, U\}$.
- When a session of Π, using the ith key as its peer key, outputs a session key, (3.15) holds for the session and it has no origin session, then:
  - If $i \neq j$, then $\mathcal{B}_2$ reveals the session key of the session and uses that key to simulate the Π^+ part of the session.
  - If $i = j$, then $\mathcal{B}_2$ tests the Π instance and uses that key to simulate the Π^+ part of the session.
- If $\mathcal{A}$ reveals the jth long-term key, $\mathcal{B}_2$ outputs 0 and stops.

If $\mathcal{A}$ breaks authentication for a session with the jth key as its peer key, $\mathcal{B}_2$ outputs 1, otherwise $\mathcal{B}_2$ outputs 0.

Let $\mathsf{Win}'_{\mathcal{B}_2,b}$ denote the event that $\mathcal{B}_2$ outputs 1, when its experiment has the secret bit b. We have that

$$\mathbf{Adv}_{\Pi,U}^{\mathsf{SelKeySecWFS}}(\mathcal{B}_2) = |\Pr[\mathsf{Win}_{\mathcal{B}_2,0}] - \Pr[\mathsf{Win}_{\mathcal{B}_2,1}]|. \tag{3.37}$$

If the experiment's secret bit $b = 0$, then $\mathcal{B}_2$ perfectly simulates Game $\mathbf{G}_2$ with respect to the $\mathsf{Win}'_{\mathbf{G}_2}$ event, since the only observable difference is that $\mathcal{B}_2$ terminates when $\mathsf{Win}'_{\mathbf{G}_2}$ no longer can occur (when the jth long-term key is revealed), so

$$\Pr[\mathsf{Win}_{\mathcal{B}_2,0}] = \Pr[\mathsf{Win}'_{\mathbf{G}_2}]. \tag{3.38}$$

If the experiment's secret bit $b = 1$, then $\mathcal{B}_2$ perfectly simulates Game $\mathbf{G}_3$ with respect to the $\mathsf{Win}'_{\mathbf{G}_3}$ event, again because of termination, so

$$\Pr[\mathsf{Win}_{\mathcal{B}_2,1}] = \Pr[\mathsf{Win}'_{\mathbf{G}_3}]. \tag{3.39}$$

The claim follows from (3.32)–(3.39).

□

3.4.3 Additional security reductions

Lemma 1 is *the* key result needed to show that protocol Π^+ achieves full forward secrecy (and explicit authentication) from the weakly forward-secret (and implicitly authenticated) protocol Π in a way that only incurs a tightness loss of U. From this result all the other security properties defined in Section 3.3 follow in a straightforward and modular way as we demonstrate in the following subsections. Moreover, none of these reductions lose more than a factor of U (in fact, most of the reductions are fully tight; those that are not only accrue the U term as a result of invoking Lemma 1).

Match soundness. Match soundness of protocol Π^+ follows directly from match soundness of Π.

Proposition 1. Let $\mathcal{A}$ be an adversary against match security for Π^+. Then there exists an adversary $\mathcal{B}$ against match security for Π with the same runtime as $\mathcal{A}$, such that

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}).$$

Proof. If any condition on the left hand side in Definition 25 (partnering via matching conversations) is satisfied for protocol Π^+ then it is also satisfied for protocol Π since the transcript of Π is a prefix of the transcript for Π^+. Hence any violation in Π^+ implies a violation in Π.

The adversary $\mathcal{B}$ against Π therefore trivially simulates an execution of Π^+ by revealing session keys and simulating the key confirmation messages. This simulation is perfect and does not require extra resources. □

Key-match soundness. Key-match soundness breaks down into two cases depending on whether or not s and s' have the same key in protocol Π. Here we again use the same-prefix security notion $\mathsf{PreEqAllEq}$ as in Lemma 1.

Proposition 2. Let $\mathcal{A}$ be an adversary against key match soundness for Π^+. Then there exists an adversary $\mathcal{B}$ against key match soundness, match security and same-prefix security for Π with the same runtime as $\mathcal{A}$, such that

$$\mathbf{Adv}^{\mathsf{KMSound}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}).$$

Proof. Suppose s and s' are such that key-match soundness (Definition 26) is violated for protocol Π^+. That is, s has accepted, $\mathsf{aFresh}(s) = \mathsf{true}$, $\mathsf{SameKey}(s, s') = \mathsf{true}$, but $\mathsf{Partner}(s, s') = \mathsf{false}$. Note that all of these predicates are relative to protocol Π^+.

We consider two cases:

1. s and s' also have the same key (k, t, t') in protocol Π;
2. s and s' do not have the same key in protocol Π.

In case 1, s and s' have the same key confirmation tags t, t', hence for them not to be partners in Π^+, they cannot be partners in protocol Π (recall that partnering is based on matching conversations). But this then violates key-match soundness of Π. In case 2, $\mathsf{PreEqAllEq}$ fails for protocol Π. □

Implicit key authentication. The proof of implicit key authentication follows the exact same structure as the key-match soundness proof.

Proposition 3. Let $\mathcal{A}$ be an adversary against implicit key authentication for Π^+. Then there exists an adversary $\mathcal{B}$ against implicit key authentication, match security and same-prefix security for Π with the same runtime as $\mathcal{A}$, such that

$$\mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}).$$

Proof. Suppose s and s' are such that implicit key authentication (Definition 27) is violated for protocol Π^+. That is, s has accepted, $\mathsf{aFresh}(s) = \mathsf{true}$, $\mathsf{SameKey}(s, s') = \mathsf{true}$, but $s.\mathsf{peer} \neq s'.\mathsf{party}$. Note that all of these predicates are relative to protocol Π^+.

We consider two cases:

1. s and s' also have the same key (k, t, t') in protocol Π;
2. s and s' do not have the same key in protocol Π.

In case 1 implicit key authentication is also violated for protocol Π. In case 2, $\mathsf{PreEqAllEq}$ fails for protocol Π. □

Explicit entity authentication. Proving explicit entity authentication is straightforward, though it relies on a technical result from Appendix C.

Proposition 4. Let $\mathcal{A}$ be an adversary against full (resp. almost-full) explicit entity authentication for Π^+. Then there exists adversaries $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_6$ against, respectively, match security, implicit key authentication, key match soundness, key secrecy, selective key secrecy and implicit key authentication for Π, all with the same runtime as $\mathcal{A}$, such that

$$\begin{aligned}\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_1) + 2 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_3) \\ & + 3 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_4) + \frac{US}{2^{\mathsf{taglen}}} + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_5)\end{aligned}$$

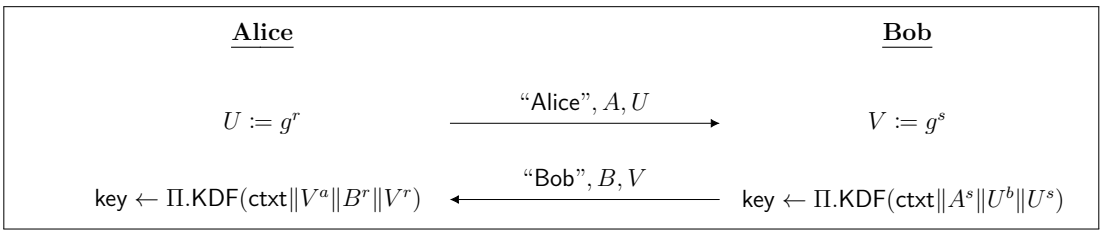

Figure 3.4: The CCGJJ protocol from [CCG+19] for a prime-ordered group $\mathbb{G}$ with generator g. Alice has secret long-term key a with public key $A \coloneqq g^a$; Bob has secret long-term key b with public key $B \coloneqq g^b$. Their context ctxt contains their names, their public keys and the two messages U and V. We include names and public keys in the messages; in practice these may be communicated in other ways.

$$\begin{aligned}\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_1) + 2 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_3) \\ + 3 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_4) + \frac{US}{2^{\mathsf{taglen}}} + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_5)\end{aligned}$$

Proof. The proofs of full and almost-full explicit entity authentication are virtually identical so we only give a proof of the former.

By Proposition 8 we have (note that Π^+ appears on both sides of the inequality)

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A})$$

Proposition 4 now follows by bounding all the individual terms on the right using, respectively, Proposition 1, Proposition 3, Proposition 2, and Lemma 1. □

3.5 The CCGJJ Protocol

The CCGJJ protocol [CCG+19], shown in Fig. 3.4, is a highly efficient implicitly authenticated key exchange protocol with optimal tightness. We use this protocol to illustrate our framework, which means we need to prove it satisfies the various security properties defined in Section 3.3.

We begin by proving that the protocol has the basic properties we want, in particular match soundness, key match soundness and the same prefix property.

Proposition 5. Let $\mathcal{A}$ be an adversary against the CCGJJ protocol. Then

$$\mathbf{Adv}^{\mathsf{Match}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|}}, \quad \mathbf{Adv}^{\mathsf{KMSound}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{|\mathbb{G}|} \quad \text{and} \quad \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|}}.$$

Proof. Since the KDF used to compute the session key includes the transcript and the secrets included are fully determined by the public values, it follows that any two sessions that are partners will compute the same session key, and a session will compute the same key as any origin session.

Match soundness could fail if a session has more than one partner, but this will only happen if two sessions choose the same randomness.

Since the information in the transcript, and in particular names, is included in the KDF that computes the key, two sessions that compute the same session key must either be partners/agree on identities or there must be a collision in the KDF.

In either case, the claims follows from a birthday bound. □

Proposition 6. Let $\mathcal{A}$ be an adversary against the CCGJJ protocol. Then

$$\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|-2\mathsf{taglen}}}.$$

Proof. Two sessions have identical prefixes but distinct keys only if the data hashed is distinct and there is a partial collision in the KDF. Since we model the KDF as a random oracle, the birthday bound applies. □

Proposition 7. Let $\mathcal{A}$ be an adversary against selective key secrecy for CCGJJ. Then there exists adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ against strong Diffie-Hellman (in group $\mathbb{G}$ with generator g) with essentially the same runtime as $\mathcal{A}$ such that

$$\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_1) + \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_3) + \frac{US^2}{|\mathbb{G}|}.$$

The proof of Proposition 7 closely follows the structure of the proof in [CCG+19], so we only sketch the argument, with emphasis on the differences, which entirely relate to avoiding hybrid arguments involving the users. (Also note that the proof in [CCG+19] proves more than just key secrecy, since they use a different security model.)

Proof (sketch). Note that all of the adversaries we construct below have the same runtime as $\mathcal{A}$ and the number of DH oracle calls is essentially bounded by the number of random oracle queries made by $\mathcal{A}$.

Recall from the proof in [CCG+19] that there are five classes of sessions: (I) initiator sessions that have an origin session/partner that it agrees on identities with; (II) other initiator sessions that are authentication fresh when they accept; (III) responder sessions that have an origin session that it agrees on identities with; (IV) other responder sessions that are authentication fresh when they accept; and (V) sessions where the intended peer's long-term key has been revealed.

The proof proceeds as a sequence of games, where the initial (zeroth) game is the selective key secrecy game. The first game prevents two honest sessions from having the same randomness. The second game changes to lazy evaluation of the random oracle $\Pi.\mathsf{KDF}$. We use the birthday bound to bound the advantage loss from the first change, while the second change is unobservable.

The third game modifies type IV responder sessions whose peer is the selected party so that it never modifies the random oracle to be consistent with the session

key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple into the public key of the selected party and (rerandomized) into the message sent by type IV responder sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_1$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to sessions running as the selected party. *Unlike in [CCG+19], the strong Diffie-Hellman adversary does not have to guess a party whose secret long-term key will not be revealed, so it does not lose a factor U in advantage.*

The fourth game modifies type III responder sessions so that they never modify the random oracle to be consistent with the session key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple (rerandomized) into the initial messages of type I and II initiator sessions and (rerandomized) into the responder messages of type III responder sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_2$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to type II initiator sessions.

The fifth game modifies type II initiator sessions whose peer is the selected party so that they never modify the random oracle to be consistent with the session key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple into the public key of the selected party and (rerandomized) into the message sent by type I or II initiator sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_3$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to sessions running as the selected party. *Again, unlike in [CCG+19], the strong Diffie-Hellman adversary does not have to guess a party whose secret long-term key will not be revealed, so it does not lose a factor U in advantage.*

At this point, we observe that every session key fresh session fails to reprogram the random oracle to be consistent with its session key, which means that every testable session key is independent of anything the adversary has observed. It follows that the adversary's advantage in this game is 0, and the claim follows. □

3.6 Impossibility of tight key confirmation

In this section we show that a large, natural, and widely-used class of compilers for turning implicitly authenticated protocols into explicitly authenticated protocols, inevitably must incur a linear security loss in the number of parties. The class includes the generic compiler from [CCG+19], which was incorrectly claimed to achieve tight security, but also the MAC-based approach to turn HMQV into HMQV-C [Kra05] (which does not give explicit security bounds) and the compiler by Yang [Yan13b] (which has a linear loss in the number of parties times sessions per party).

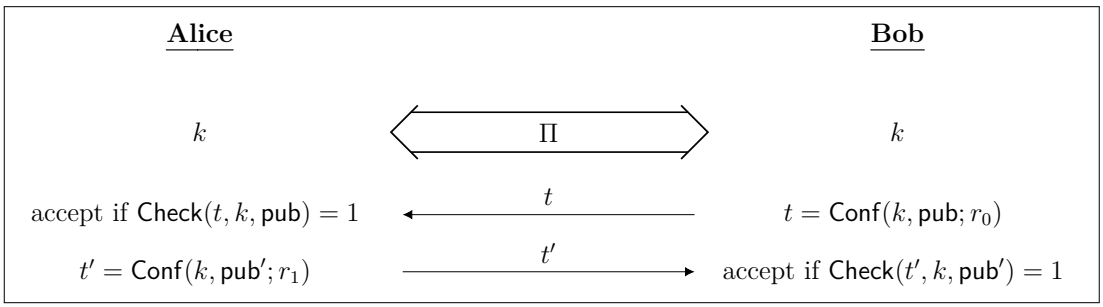

Figure 3.5: Protocol Π^+ obtained by extending protocol Π with key confirmation tags t, t'. The session sending the last message in protocol Π sends tag t, and the session receiving the last message in protocol Π sends tag t'. pub contains the transcript of Π and public keys $\mathsf{pk}_A, \mathsf{pk}_B$. pub' additionally contains the first key confirmation tag t.

3.6.1 Requirements on Π and Π^+

We consider generic compilers that turn implicitly authenticated protocols Π with weak forward security into explicitly authenticated protocols Π^+ with full forward security. To this end, we will in the sequel focus on underlying protocols Π and constructions Π^+ that satisfy certain requirements that we define in this section.

Key confirmation messages. We assume an n-message protocol Π is extended to an $(n+1)$-message protocol Π^+ as shown in Figure 3.5. The participants first run the protocol Π and then exchange two key confirmation message t, t', where the first key confirmation message t is sent together with the final message of Π and t' in reply as the final $(n+1)$-th message. Alternatively, we could define Π^+ such that the first key confirmation message is sent as a *reply* to the n-th message. This would add *two* messages to Π, making Π^+ an $(n+2)$-message protocol. We consider the former approach more natural, and this is the approach used in CCGJJ19 [CCG+19] and HMQV-C [Kra05]. The latter approach is used by Yang [Yan13b]. Even though we only treat the former variant here, our results apply equally to both variants.

The parties compute the key confirmation messages with a potentially non-deterministic algorithm $\mathsf{Conf}\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$, where $\mathcal{K}$ is the session key space of Π. We require that apart from the session key, t, t' are computed only from publicly known values, such as the protocol transcript or public keys. In practice, Conf typically is a PRF+simple MAC construction computed over the protocol transcript (as e.g. in [CCG+19]).

To verify the key confirmation messages we assume a deterministic algorithm $\mathsf{Check}\colon \{0,1\}^\tau \times \mathcal{K} \times \{0,1\}^* \to \{0,1\}$. We require that the probability of a given tag being valid under a randomly chosen key is negligible. Formally, we need that they are *δ-entropy-preserving* in the following sense.

Definition 33. We say that an algorithm Check is *δ-entropy-preserving* if for all $k \in \mathcal{K}, \mathsf{pub} \in \{0,1\}^*$, and all $t \xleftarrow{\$} \mathsf{Conf}(k, \mathsf{pub})$ we have

$$\Pr_{k' \xleftarrow{\$} \mathcal{K}}[\mathsf{Check}(t, k', \mathsf{pub}) = 1] \leq \delta.$$

Security of Π^+ from Π via black-box reduction. We consider generic constructions, where the security of Π^+ is based on the security of Π and possibly the security of the primitives used to generate the key confirmation messages. This excludes artificial constructions of Π^+, which run Π as a redundant subroutine, but where security is achieved in a completely different way, such that Π is actually superfluous. The most natural way to establish security of Π^+ based on the security of Π is a reduction $\mathcal{R}$ from, e.g., the $\mathsf{KeySecFFS}$ security of Π^+ to the $\mathsf{KeySecWFS}$ of Π and potentially other arguments and reductions to the security of primitives used in Π^+.

We assume that $\mathcal{R}$ is given black-box access to an adversary $\mathcal{A}$ that breaks the $\mathsf{KeySecFFS}$ security of Π^+ and interacts with $\mathcal{A}$ via the oracles defined in the $\mathsf{KeySecFFS}$ security experiment given in Section 3.2. At the same time $\mathcal{R}$ has access to a $\mathsf{KeySecWFS}$ security experiment for Π. We assume that $\mathcal{R}$ is able to win that security experiment with non-neglible probability if $\mathcal{A}$ has a non-neglible probability of winning the security experiment for Π^+. We only consider black-box reductions which run a single instance of $\mathcal{A}$ once, without rewinding.

Furthermore, we require that $\mathcal{R}$ is "valid" in the sense that it never makes queries that would lead to $\mathcal{R}$ trivially losing the security experiment, such as issuing both $\textsc{Test}(s)$ and $\textsc{Reveal}(s)$ for some session s. Note that any reduction $\mathcal{R}$ that does not satisfy this assumption can be turned into a reduction $\mathcal{R}'$ that satisfies the assumption by relaying all queries from $\mathcal{R}$ until $\mathcal{R}$ makes an invalid query and then simply aborting. Similarly, we assume that $\mathcal{R}$ never ignores oracle queries by $\mathcal{A}$ and always outputs some reply, which, however, may potentially be invalid.

Finally, we assume that for every party i in the security experiment for Π^+ there exists a unique corresponding party i' in the security experiment for Π simulated by $\mathcal{R}$ towards $\mathcal{A}$ with $\mathsf{pk}_i = \mathsf{pk}_{i'}$. Note that this does not exclude reductions that run the adversary against Π^+ with a smaller number of users than they were given from the experiment on Π. We only require that $\mathcal{R}$ did not generate any public key given to $\mathcal{A}$ by itself.

Π has unique and efficiently verifiable secret keys. We assume that for each public key pk there is a unique matching secret key and that it is possible to efficiently and perfectly verify whether a given secret key sk matches a given public key pk. For Diffie–Hellman-like protocols such as HMQV [Kra05] or NAXOS [LLM07], where public keys are typically of the form g^x with x being the secret key, this assumption trivially holds. For many other protocols, secret keys can often be verified via executing a full protocol session with the given secret key and some

known key pair belonging to some other party and checking whether both parties compute the same session key. As mentioned above, one can easily generalize our result from unique to efficiently *rerandomizable* keys, using techniques from [HJK12, BJLS16]. Essentially, the rerandomization ensures that a meta-reduction can turn a given secret key for public key pk into a uniformly random secret key among all valid secret keys for pk. We omit this, since this is a relatively straightforward generalization.

3.6.2 Impossibility result

We first describe an *inefficient hypothetical* adversary $\mathcal{A}$ against the protocol Π^+ that breaks the $\mathsf{KeySecFFS}$ security with almost certainty. We then argue that for any *tight* reduction $\mathcal{R}$ from the $\mathsf{KeySecFFS}$ security of Π^+ to the $\mathsf{KeySecWFS}$ security of Π we are able to *efficiently* simulate the hypothetical adversary $\mathcal{A}$ in a meta-reduction $\mathcal{M}$. This means given a tight reduction $\mathcal{R}$, $\mathcal{M}$ is able to effciently break the $\mathsf{KeySecWFS}$ security of Π. This is a contradiction to the assumption that Π is secure and implies that $\mathcal{R}$ cannot be tight. We visualize the meta-reduction $\mathcal{M}$ in Figure 3.6.

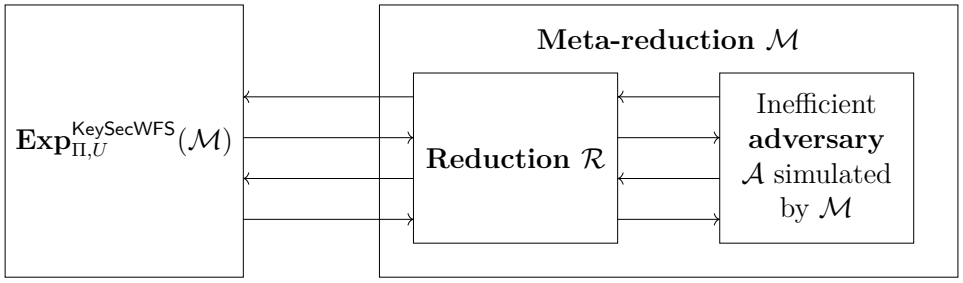

Figure 3.6: Visualization of meta-reduction $\mathcal{M}$ playing the experiment $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{M})$ while running $\mathcal{R}$ as a subroutine and simulating the inefficient adversary $\mathcal{A}$ to $\mathcal{R}$. $\mathcal{M}$ relays all queries from $\mathcal{R}$ to $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{M})$.

Intuition. Let us sketch the main idea of the construction of our hypothetical adversary and its simulation in the meta-reduction. For each party i, it creates a new session s_i, where it impersonates party i towards party $i+1$, and lets the reduction $\mathcal{R}$ compute the first key confirmation message in s_i. Afterwards, $\mathcal{A}$ invalidates the freshness of all sessions except one randomly chosen session s_{i^*} by corrupting all parties except i^*+1. This essentially forces $\mathcal{R}$ to correctly guess the value i^* before computing the key confirmation messages, as it cannot leverage any session except s_{i^*} to break the security of Π. Since i^* was chosen uniformly at random by $\mathcal{A}$, $\mathcal{R}$ can only guess correctly with probability $1/U$, which implies that $\mathcal{R}$ cannot be tight and must lose a factor U.

However, in order to efficiently simulate $\mathcal{A}$, $\mathcal{M}$ must be able to compute messages of Π on behalf of any party. A significant challenge in our result is that these messages may depend on a secret key unknown to $\mathcal{M}$. Furthermore, $\mathcal{M}$ cannot corrupt a party to obtain its secret key and compute the messages of Π, since corrupting all parties except i^* *after* $\mathcal{R}$ already produced a valid key confirmation message for the parties is a crucial tool in forcing $\mathcal{R}$ to guess i^*.

To overcome this challenge, we let the security experiment for Π generate the messages in newly created sessions. This requires carefully arguing that the existence of these sessions can be hidden from $\mathcal{R}$ and they cannot influence the validity of any query $\mathcal{R}$ may make such that $\mathcal{M}$ can still perfectly simulate the experiment for Π towards $\mathcal{R}$.

Strictly speaking, $\mathcal{M}$ will use twice as many Π sessions as $\mathcal{A}$, but this remains tight, and regardless, $\mathcal{A}$ uses just as many sessions as there are users, which is a trivial number of sessions. Furthermore, in order to rule out a tighter reduction, it is sufficient to decribe an arbitrary, polynomial-time meta-reduction that shows how to efficiently (*i.e.*, in polynomial time) solve a computationally infeasible problem, such as breaking the underlying protocol, with non-negligible advantage. We do not need the meta-reduction to be tight.

Theorem 2. *Let* Π *be an AKE protocol and* Π^+ *an AKE protocol constructed by extending* Π *with key confirmation using* Conf *and* Check *as described in Section 3.6.1. Let* $\mathcal{R}$ *be a reduction that uses any adversary* $\mathcal{A}$ *against the* $\mathsf{KeySecFFS}$ *security of* Π^+ *to break the* $\mathsf{KeySecWFS}$ *security of* Π *and* Conf *and* Check *be* δ*-entropy-preserving. If* Π, Π^+*, and* $\mathcal{R}$ *satisfy the requirements listed above in Section 3.6.1, we can construct efficient adversaries* $\mathcal{M}, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ *with*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M}) \geq \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{R}(\mathcal{A})) - 3 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{B}_1) \\ - \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U'}(\mathcal{B}_2) - \sqrt{\mathbf{Adv}^{\mathsf{Match}}_{\Pi,U'}(\mathcal{B}_3)} - 2 \cdot (\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{B}_4) + \frac{1}{U} - U\delta)\end{aligned}$$

Interpretation of Theorem 2. We can assume that δ and the advantages of the adversaries $\mathcal{B}_1$, $\mathcal{B}_2, \mathcal{B}_3$ and $\mathcal{B}_4$ are negligible. Further, if $\mathcal{R}$ was tight, its advantage would be close to the advantage of $\mathcal{A}$. The adversary $\mathcal{A}$ we describe below breaks the $\mathsf{KeySecFFS}$ security of Π^+ with probability very close to 1 and thus $\mathcal{M}$ would break the $\mathsf{KeySecWFS}$ security of Π with probability roughly $1 - 2/U$. However, this is a contradiction to the assumption that Π is a secure protocol and implies that $\mathcal{R}$ cannot be tight. In particular, in order for the advantage of $\mathcal{M}$ being negligible, $\mathcal{R}$ must lose a factor $U/2$.

Proof. We begin by describing a hypothetical adversary $\mathcal{A}$ that *inefficiently* breaks the security of Π^+ with high probabilty. Given any reduction $\mathcal{R}$, we then construct a meta-reduction $\mathcal{M}$ that *efficiently* simulates $\mathcal{A}$ if $\mathcal{R}$ is a tight reduction. This means that by using a tight reduction $\mathcal{R}$, the meta-reduction $\mathcal{M}$ is efficiently able to break the security of Π. This is a contradiction to the assumption that Π is secure and implies that $\mathcal{R}$ cannot be tight.

Hypothetical adversary $\mathcal{A}$. Consider the following hypothetical (inefficient) adversary $\mathcal{A}$ against the $\mathsf{KeySecFFS}$ security of Π^+.

1. $\mathcal{A}$ receives a list of public keys $\mathbf{pk}$ from its experiment and chooses $j^* \xleftarrow{\$} \{1, \ldots, U\}$ at random. Let $i^* := j^* - 1 \bmod U$.

2. For all $i \in \{1, \ldots, U\}$, $\mathcal{A}$ initiates a new session owned by i with the intended peer $j = i + 1 \bmod U$, where the role of i depends on the number of messages in Π. Let n be the number of messages in Π. If n is even, $\mathcal{A}$ chooses the role resp, and otherwise the role init. This ensures that in this new session, the party that sends the last message of Π and hence also the first key confirmation message is the party i. Formally, $\mathcal{A}$ issues the query $\textsc{NewSession}(i, j, \mathsf{role})$ with role set as described above and in return gets a session number $s_{i,j}$ and potentially the initial message $m_{i,0}$.

3. For all sessions $s_{i,j}$ with $i \in \{1, \ldots, U\}$ and $j = i + 1 \bmod U$, $\mathcal{A}$ executes Π^+ impersonating the party j until the experiment outputs the first key confirmation message t_i. To this end, it first runs $(st_j, m_{i,0}) \xleftarrow{\$} \mathsf{Init}(j, i, \mathsf{sk}_j, \mathsf{pk}_i, \mathsf{role})$, where $\mathsf{role} = \mathsf{init}$ if n is even and $\mathsf{role} = \mathsf{resp}$ otherwise. Next, if the role of j is init, $\mathcal{A}$ sends the initial message on behalf of j by querying $\textsc{Send}(s_{i,j}, m_{i,0})$ and obtains as output the response $m_{i,1}$. If $m_{i,1}$ does not contain the first key confirmation message, $\mathcal{A}$ computes the next protocol message $m_{i,2}$ as $(st_j, m_{i,2}) \xleftarrow{\$} \mathsf{Run}(st_j, m_{i,1})$ and sends it to the session $s_{i,j}$. $\mathcal{A}$ continues this process until the session $s_{i,j}$ outputs the first key confirmation message t_i. If the role of j is resp, $\mathcal{A}$ acts analogously, except that it uses the initial message $m_{i,0}$ it obtained in Step 2 to compute the first message of j. Note that even if any protocol message that $\mathcal{A}$ needs to compute depends on the secret key sk_j, $\mathcal{A}$ is able to (inefficiently) compute the message by, e.g., brute forcing sk_j.

4. $\mathcal{A}$ issues a query $\textsc{RevealLTK}(i)$ for all $i \neq j^*$. Due to the assumption that Π has efficiently and perfectly verifiable secret keys, $\mathcal{A}$ can somehow check whether all secret keys returned by the experiment are correct. If any secret key does not match the corresponding public key, $\mathcal{A}$ aborts.

5. For all $i \in \{1, \ldots, U\}$, $\mathcal{A}$ verifies that the key confirmation message t_i is valid and aborts if it is not valid. Again, note that $\mathcal{A}$ is somehow able to (inefficiently) compute the sesion key k_i that it needs to run $\mathsf{Check}(\mathsf{pub}_i, k_i, t_i)$.

6. $\mathcal{A}$ somehow computes the second key confirmation message t'_{i^*} and sends it to the session s_{i^*,j^*} with the query $\textsc{Send}(s_{i^*,j^*}, t'_{i^*})$. Finally, $\mathcal{A}$ issues the query $\textsc{Test}(s_{i^*,j^*})$ to obtain a challenge key k. If $k = k_{i^*}$, $\mathcal{A}$ outputs $b = 0$ and otherwise $b = 1$.

Let us now analyze the advantage of $\mathcal{A}$. Note that for the session s_{i^*,j^*} we have $\mathsf{aFresh}(s_{i^*,j^*}) = \mathsf{true}$ since $\mathcal{A}$ did not corrupt the party j^*. Hence, even

though there is no origin session for s_{i^*,j^*}, $\mathsf{kFreshFFS}(s_{i^*,j^*})$ holds. This means the hypothetical $\mathcal{A}$ breaks the $\mathsf{KeySecFFS}$ predicate with probability $1 - 1/|\mathcal{K}|$ and we have

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) = 1 - \frac{1}{|\mathcal{K}|},$$

where $1/|\mathcal{K}|$ is deducted due to the probability of TEST returning a randomly chosen key that is equal to the real session key.

Construction of meta-reduction $\mathcal{M}$. Let $\mathcal{R}$ be a reduction as described in Theorem 2. Recall that $\mathcal{R}$ is a black-box reduction that should be successful against the security of Π given any successful adversary against the security of Π^+. In particular, it should be successful with our hypothetical adversary $\mathcal{A}$. We now describe our meta-reduction $\mathcal{M}$ that uses $\mathcal{R}$ as a subroutine while simulating $\mathcal{A}$ towards $\mathcal{R}$. $\mathcal{M}$ plays the security experiment $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ and relays queries between its experiment and $\mathcal{R}$. $\mathcal{M}$ simulates $\mathcal{A}$ using the following steps (see also Figure 3.7).

1. $\mathcal{M}$ receives a list of public keys $\mathbf{pk}'$ of size U' from its experiment and provides it to $\mathcal{R}$. $\mathcal{R}$ then outputs a list of public keys $\mathbf{pk}$ of size U with $U \leq U'$. Exactly as $\mathcal{A}$, $\mathcal{M}$ chooses $j^* \xleftarrow{\$} \{1, \ldots, U\}$ at random, and again let $i^* := j^* - 1$.

   As described in Section 3.6.1, we assume that for each party i in the experiment $\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A})$ simulated by $\mathcal{R}$ there exists a unique corresponding party i' in the experiment $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\lambda)$ with $\mathsf{pk}_i = \mathsf{pk}_{i'}$. For simplicity, we assume in the following that $\mathcal{R}$ does not modify $\mathbf{pk}$ when forwarding it to its adversary, i.e., we have $i = i'$ and $U = U'$. If $\mathcal{R}$ applies some permutation π to $\mathbf{pk}'$ or deletes some entries before outputting it again, $\mathcal{M}$ can simply compare the lists of public keys to compute the inverse permutation π^{-1}, which then maps any party i in $\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A})$ to the corresponding party i' in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$.

2. This step is exactly the same as Step 2 of $\mathcal{A}$.

3. This is the first hypothetical step of $\mathcal{A}$, which $\mathcal{M}$ needs to efficiently simulate. While $\mathcal{A}$ can, e.g., simply brute force the secret key of some party j to impersonate j and compute any messages that may depend on sk_j, this is not possible for $\mathcal{M}$. Instead, we rely on our assumption that for any party in the experiment for Π^+ simulated by $\mathcal{R}$ there exists a corresponding party in the experiment for Π, which allows us to let $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ generate all messages on behalf of any party in $\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A})$.

   For this, $\mathcal{M}$ proceeds as follows. Similarly to the previous step, $\mathcal{M}$ creates a new session in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ for each party $j \in \{1, \ldots, U\}$ with the intended peer $i = j - 1$, however with the role chosen in converse. If the

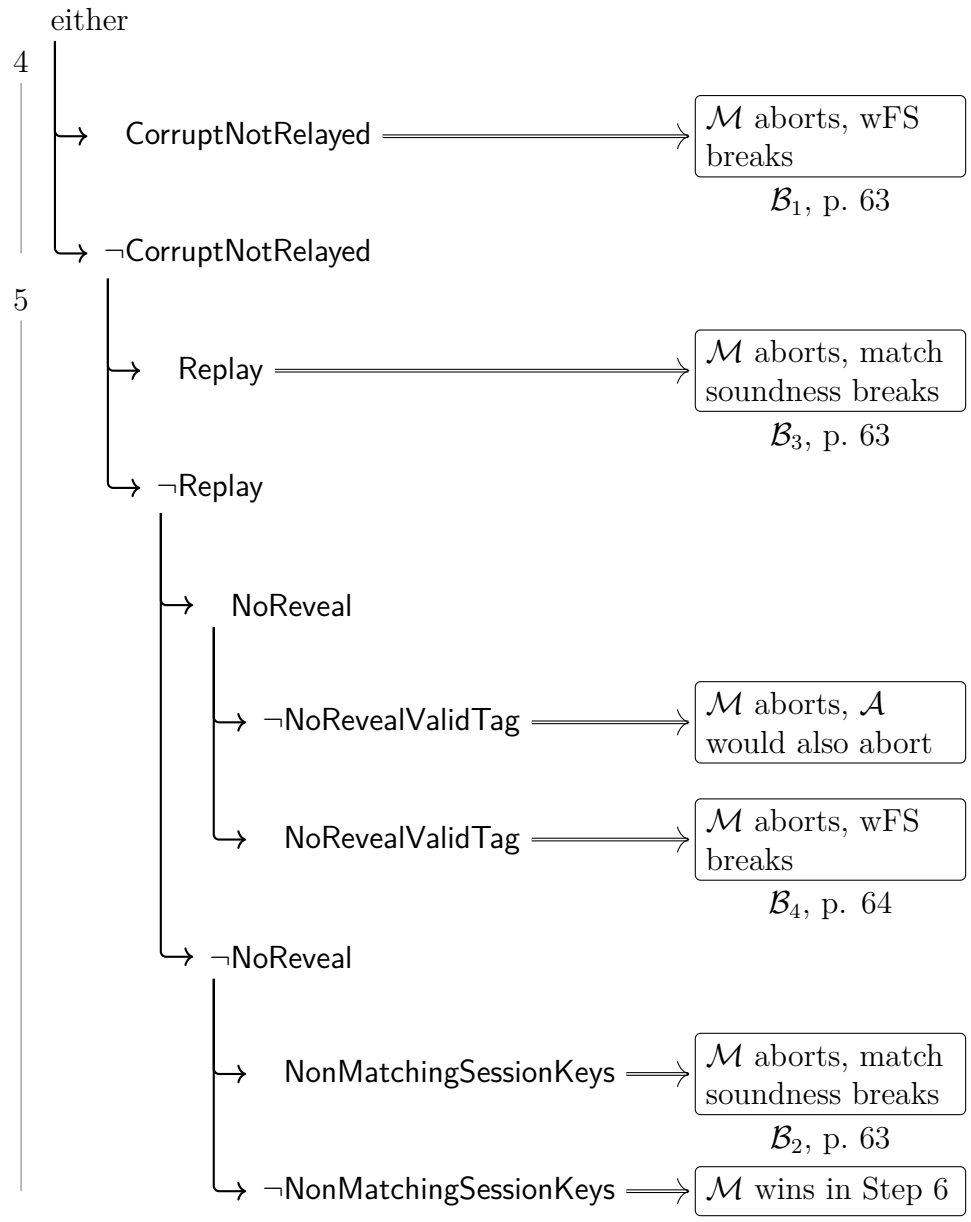

Figure 3.7: The meta-reduction Steps 1–3 start sessions such that the reduction sends key confirmation tags, Step 4 corrupts all but one long-term key, Step 5 verifies the tags and Step 6 sends a correct tag for the randomly chosen session. The main difficulty is the analysis of Steps 4 and 5. The diagram describes the case analysis tree for Steps 4 and 5 of the meta-reduction $\mathcal{M}$. Labels written in sans serif font are events that can happen in Steps 4 and 5 of $\mathcal{M}$, with $\Rightarrow$ pointing to the corresponding action taken by $\mathcal{M}$.

number of messages n in Π is even, it issues the query NewSession(j, i, init) to its experiment and otherwise the query NewSession(j, i, resp). $\mathcal{M}$ here essentially creates the opposing peer sessions $\tilde{s}_{j,i}$ for all sessions $s_{i,j}$ it created in the previous step. For readability, in the following we denote any session in $\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A})$ with s and any session in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ with $\tilde{s}$.

After creating the new sessions in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$, $\mathcal{M}$ is able to simulate $\mathcal{A}$ by relaying the messages between $\mathcal{R}$ and $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ with the respective Send queries. First assume that n is even. In that case, for all sessions $\tilde{s}_{j,i}$ created in this step we have $\mathsf{role} = \mathsf{init}$ and in response to the NewSession queries $\mathcal{M}$ obtains the first message $m_{i,0}$. Then, $\mathcal{M}$ alternates between issuing Send queries to $\mathcal{R}$ for the session $s_{i,j}$ and to $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ for the session $\tilde{s}_{j,i}$ until $\mathcal{R}$ outputs the first key confirmation message. The first Send query is given to $\mathcal{R}$ with the message $m_{i,0}$ and any subsequent query is given with the message output by the previous Send query. If n is odd, $\mathcal{M}$ obtained the first message $m_{i,0}$ already in Step 2 and issues its first Send query to the experiment instead of $\mathcal{R}$. After $\mathcal{R}$ outputs the first key confirmation message t_i together with the final message $m_{i,n-1}$ of Π, $\mathcal{M}$ sends $m_{i,n-1}$ to the session $\tilde{s}_{j,i}$ such that it accepts.

Note that $\mathcal{M}$ needs to keep the existence of the sessions $\tilde{s}_{j,i}$ hidden from $\mathcal{R}$ in order to perfectly simulate the security experiment for $\mathcal{R}$. In particular, $\mathcal{M}$ needs to decrease the session number returned by its experiment in response to any subsequent NewSession query relayed from $\mathcal{R}$ by U and ensure that the existence of the hidden sessions and any queries issued to them do not disallow any potential queries from $\mathcal{R}$.

As all messages output by $\mathcal{M}$ in this step are computed by the experiment, the message distribution is obviously identical to the distribution of messages output by $\mathcal{A}$, which computed the messages according to Π. Thus, $\mathcal{M}$ perfectly simulates $\mathcal{A}$ in this step.

4. $\mathcal{M}$ first executes all instructions from Step 4 of $\mathcal{A}$. Additionally, we denote by $\mathsf{CorruptNotRelayed}$ the event that $\mathcal{R}$ answers a RevealLTK query with a correct secret key without ever having given the same query to its experiment. If $\mathsf{CorruptNotRelayed}$ occurs, we let $\mathcal{M}$ abort. Recall that $\mathcal{M}$ runs $\mathcal{R}$ as a subroutine and relays all queries from $\mathcal{R}$ to its experiment, which allows it to observe whether $\mathsf{CorruptNotRelayed}$ occurs. Therefore, if $\mathsf{CorruptNotRelayed}$ does not occur, $\mathcal{M}$ simulates $\mathcal{A}$ in this step. We defer bounding $\Pr[\mathsf{CorruptNotRelayed}]$ to later (p. 63).

5. To check the validity of all tags t_i, $\mathcal{M}$ needs to know the session key k_i established in the session $\tilde{s}_{j,i}$. If the session $\tilde{s}_{j,i}$ does not have a partner in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$, which may happen if $\mathcal{R}$ computes the messages of the session $s_{i,j}$ without creating a corresponding session $\tilde{s}_{i,j}$ in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$, $\mathcal{M}$ can simply issue the query Reveal$(\tilde{s}_{j,i})$.

Here, we need to ensure that revealing a session $\tilde{s}_{j,i}$ that has no partner does not restrict the queries that $\mathcal{R}$ is allowed to make. As long as $\tilde{s}_{j,i}$ is not partnered, revealing it does not influence the session key freshness of any other session. However, if $\tilde{s}_{j,i}$ ever becomes partnered with some session $\tilde{s}$ at some later point, which we denote as the event Replay, $\tilde{s}$ is not session key fresh. But since $\tilde{s}_{j,i}$ is hidden from $\mathcal{R}$, from the view of $\mathcal{R}$ the session $\tilde{s}$ does not have partner and therefore should be session key fresh. In that case $\mathcal{M}$ cannot simulate the experiment for $\mathcal{R}$ anymore. We bound the probability of Replay occurring below (see p. 63).

If during this step $\tilde{s}_{j,i}$ is already partnered with some session $\tilde{s}$ that $\mathcal{R}$ may have created during Step 3 by relaying all SEND queries for the session $s_{i,j}$ to its experiment, revealing $\tilde{s}_{j,i}$ would invalidate the freshness of $\tilde{s}$. This means that if $\mathcal{R}$ would then test the session $\tilde{s}$, $\mathcal{M}$ cannot simulate the experiment for Π towards $\mathcal{R}$ anymore. Thus, if the session $\tilde{s}_{j,i}$ is partnered with some other session $\tilde{s}$, $\mathcal{M}$ has to act differently and proceeds as follows.

First, let $\mathsf{Partnered}$ denote the set of sessions $(\tilde{s}_{j,i}, \tilde{s})$, where $\mathsf{Partner}(\tilde{s}_{j,i}, \tilde{s})$ holds. Further, let $\mathsf{NoReveal}$ denote the event that there exist some $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$, where $\tilde{s}$ is not revealed. If $\mathsf{NoReveal}$ does not occur, then for all $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$, $\mathcal{R}$ revealed $\tilde{s}$ as due to $\mathsf{CorruptNotRelayed}$ not occurring, the reduction cannot query $\textsc{Test}(\tilde{s})$. Since $\mathcal{M}$ can observe the experiment's response to any $\textsc{Reveal}$ query from $\mathcal{R}$, it learns the session key of the session $\tilde{s}$. However, we still need to ensure that the session $\tilde{s}_{j,i}$ actually computes the same key as $\tilde{s}$. For this, let $\mathsf{NonMatchingSessionKeys}$ denote the event that for some $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$ it holds that $\tilde{s}_{j,i}.\mathsf{key} \neq \tilde{s}.\mathsf{key}$. Then, if $\mathsf{NonMatchingSessionKeys}$ does not occur, $\mathcal{M}$ learns the correct session key of $\tilde{s}_{j,i}$ by observing the $\textsc{Reveal}$ query of $\mathcal{R}$.

If $\mathsf{NoReveal}$ occurs, we let $\mathcal{M}$ abort. Note that if $\mathsf{NoReveal}$ occurs, $\mathcal{M}$ still simulates $\mathcal{A}$ if for all $\tilde{s}_{j,i}$, which cause the event $\mathsf{NoReveal}$, the tag t_i output in the session $s_{i,j}$ is not valid as $\mathcal{A}$ aborts as well in that case. Therefore, let $\mathsf{NoRevealValidTag}$ denote the event that for some session $s_{i,j}$ the session $\tilde{s}_{j,i}$ causes the event $\mathsf{NoReveal}$ and the tag t_i is valid.

Overall we have, that $\mathcal{M}$ can simulate Step 5 of $\mathcal{A}$ independent of whether the sessions $\tilde{s}_{j,i}$ have a partner or not if the events $\mathsf{NonMatchingSessionKeys}$, $\mathsf{NoRevealValidTag}$ and Replay do not occur. We again defer bounding $\Pr[\mathsf{NonMatchingSessionKeys}], \Pr[\mathsf{NoRevealValidTag}]$ and $\Pr[\mathsf{Replay}]$ to later (p. 63).

6. If $(\tilde{s}_{j^*,i^*}, \cdot) \notin \mathsf{Partnered}$, $\mathcal{M}$ already revealed $\tilde{s}_{j^*,i^*}$ in Step 5 and obtained the session key k_{i^*}, which it can use to compute the tag $t'_{i^*} = \mathsf{Conf}(k_{i^*}, \mathsf{pub}')$, where pub' contains the transcript of the session $\tilde{s}_{j^*,i^*}$, the public keys $\mathsf{pk}_{i^*}, \mathsf{pk}_{j^*}$, and t_{i^*}. If for some session $\tilde{s}$ it holds that $(\tilde{s}_{j^*,i^*}, \tilde{s}) \in \mathsf{Partnered}$, $\mathcal{R}$ queried $\textsc{Reveal}(\tilde{s})$ since $\mathsf{NoReveal}$ did not occur. Therefore, $\mathcal{M}$ can use the key k_{i^*} returned in that $\textsc{Reveal}$ query to compute t'_{i^*} as above.

$\mathcal{M}$ then proceeds exactly as $\mathcal{A}$, i.e., it sends t'_{i^*} to the session s_{i^*,j^*}, tests s_{i^*,j^*}, compares the key k_{i^*} it obtained as described above with the challenge key k, and outputs the bit b accordingly.

In summary, $\mathcal{M}$ efficiently simulates $\mathcal{A}$ if the events NonMatchingSessionKeys, CorruptNotRelayed, Replay, and NoRevealValidTag do not occur and we have

$$\begin{aligned}\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{M}) \geq \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{R}(\mathcal{A})) - \Pr[\mathsf{CorruptNotRelayed}] \\ - \Pr[\mathsf{Replay}] - \Pr[\mathsf{NonMatchingSessionKeys}] - \Pr[\mathsf{NoRevealValidTag}].\end{aligned}$$

Hence, it remains to bound $\Pr[\mathsf{CorruptNotRelayed}], \Pr[\mathsf{NonMatchingSessionKeys}]$, $\Pr[\mathsf{Replay}]$, and $\Pr[\mathsf{NoRevealValidTag}]$.

Bounding $\Pr[\mathsf{CorruptNotRelayed}]$**.** We construct an adversary $\mathcal{B}_1$ that breaks the KeySecWFS security of Π whenever CorruptNotRelayed occurs. $\mathcal{B}_1$ acts like $\mathcal{M}$ for Steps 1-4. If CorruptNotRelayed does not occur, $\mathcal{B}_1$ outputs a random bit. Otherwise, let $\textsc{RevealLTK}(i)$ be the first query that causes CorruptNotRelayed and sk_i the key returned by $\mathcal{R}$ in that query. $\mathcal{B}_1$ chooses $j \neq i$ at random and creates a new session $\tilde{s}$ owned by j with the intended peer i in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$. Note that $\mathcal{B}_1$ is able to compute all messages on behalf of party i as it obtained sk_i from $\mathcal{R}$. Therefore, $\mathcal{B}_1$ can impersonate i throughout the session and can make the session $\tilde{s}$ accept with the appropiate $\textsc{Send}$ queries.

After $\tilde{s}$ has accepted, $\mathcal{B}_1$ tests $\tilde{s}$ and is able to distinguish the challenge key since it impersonated i throughout the session. Note that $\tilde{s}$ is authentication fresh since $\textsc{RevealLTK}(i)$ was never queried to the experiment and hence $\tilde{s}$ is also session key fresh. Thus, we have

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{B}_1) = \Pr[\mathsf{CorruptNotRelayed}].$$

Bounding $\Pr[\mathsf{NonMatchingSessionKeys}]$**.** We now construct an adversary $\mathcal{B}_2$ against the Match soundness of Π whenever the event NonMatchingSessionKeys occurs. The adversary $\mathcal{B}_2$ acts just like $\mathcal{M}$ for Steps 1 to 5 and then stops. If NonMatchingSessionKeys occurs, then there exist some $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$ with $\tilde{s}_{j,i}.\mathsf{key} \neq \tilde{s}.\mathsf{key}$, which implies that condition (3.9) does not hold and we have

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi,U'}(\mathcal{B}_2) = \Pr[\mathsf{NonMatchingSessionKeys}].$$

Bounding $\Pr[\mathsf{Replay}]$**.** We construct an adversary $\mathcal{B}_3$ that breaks the match soundness of Π whenever the event Replay occurs. $\mathcal{B}_3$ acts like $\mathcal{M}$ until the event Replay occurs and let $\tilde{s}$ be the session that causes Replay by becoming partnered with some session $\tilde{s}_{j,i}$. When Replay occurs, $\mathcal{B}_3$ stops $\mathcal{R}$ and creates a new session $\tilde{s}'$ in $\mathbf{Exp}^{\mathsf{Match}}_{\Pi,U'}(\lambda)$ with the same owner, peer, and role as $\tilde{s}$. Let $\tilde{s}.\mathsf{transcript}^{\mathsf{r}}$ be the list of messages that $\tilde{s}$ received. $\mathcal{B}_3$ sends all messages in $\tilde{s}.\mathsf{transcript}^{\mathsf{r}}$ to $\tilde{s}'$ with the appropiate $\textsc{Send}$ queries. Then $\tilde{s}'$ outputs the same messages as $\tilde{s}$ with

probability at least Pr[Replay]. If $\tilde{s}'$ outputs the same messages, $\tilde{s}_{j,i}.\tilde{s}$, and $\tilde{s}'$ are all partnered, which implies that condition (3.10) does not hold and we have

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi,U'}(\mathcal{B}_3) \geq \Pr[\mathsf{Replay}]^2,$$

as the probability that $\tilde{s}$ becomes partnered with $\tilde{s}_{j,i}$ is Pr[Replay] and the probability that $\tilde{s}$ becomes partnered with $\tilde{s}'$ is at least Pr[Replay] as well.

Bounding Pr[NoRevealValidTag]**.** We construct an adversary $\mathcal{B}_4$ that breaks the KeySecWFS security of Π whenever the event NoRevealValidTag occurs. $\mathcal{B}_4$ acts like $\mathcal{M}$ for Steps 1-5. Then, for all $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$ such that $\tilde{s}$ was not revealed, $\mathcal{B}_4$ queries $\textsc{Test}(\tilde{s}_{j,i})$ to $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$ to obtain a challenge key k_i. Next, it runs $\mathsf{Check}(t_i, k_i, \mathsf{pub}_i)$, where pub_i contains the transcript of $\tilde{s}_{j,i}$ and the public keys $\mathsf{pk}_i, \mathsf{pk}_j$, to check the tag t_i that was output by $\mathcal{R}$ in the session $s_{i,j}$. If for any $\tilde{s}_{j,i}$ the tag t_i is valid under k_i, $\mathcal{B}_4$ outputs $b = 0$ to its experiment and otherwise $b = 1$. If the event NoReveal does not occur, which implies that no session as described above exists, $\mathcal{B}_4$ outputs a random bit.

Observe that for all sessions $\tilde{s}_{j,i}$ that $\mathcal{B}_4$ tests we have $\mathsf{kFreshWFS}(\tilde{s}_{j,i}) = \mathsf{true}$ due to the following. First, the only Reveal queries that $\mathcal{B}_4$ issues are queries it relays from $\mathcal{R}$ and queries it issues in Step 5. But the session $\tilde{s}_{j,i}$ is hidden from $\mathcal{R}$ and cannot be revealed by $\mathcal{R}$ and in Step 5 $\mathcal{B}_4$ only reveals sessions that do not have a partner, which implies that condition (3.5) holds. Second, all sessions $\tilde{s}_{j,i}$ tested by $\mathcal{B}_4$ have a partner session, which is then also the origin session for $\tilde{s}_{j,i}$, and thus condition (3.7) holds.

Third, $\mathcal{B}_4$ only tests sessions $\tilde{s}_{j,i}$, where the partner session $\tilde{s}$ was not revealed by $\mathcal{R}$. Additionally, if in Step 4, CorruptNotRelayed does not occur, then all parties $j \neq j^*$ were corrupted by $\mathcal{R}$. This means for all $\tilde{s}_{j,i} \neq \tilde{s}_{j^*,i^*}$ tested by $\mathcal{B}_4$, testing the session $\tilde{s}$ partnered with $\tilde{s}_{j,i}$ would violate session key freshness of $\tilde{s}$ as condition (3.7) would not hold. We therefore ensure that $\mathcal{R}$ does not test the session $\tilde{s}$ partnered with $\tilde{s}_{j,i}$ due to the assumption that $\mathcal{R}$ does not make any invalid queries. Thus, for $\tilde{s}_{j,i} \neq \tilde{s}_{j^*,i^*}$ condition (3.6) holds.

However, since the party j^* was not corrupted, there is no guarantee that $\mathcal{R}$ did not test the session partnered with $\tilde{s}_{j^*,i^*}$. To this end, let TargetTested denote the event, that $\tilde{s}_{j^*,i^*}$ is partnered with some session $\tilde{s}$ and $\mathcal{R}$ issued the query $\textsc{Test}(\tilde{s})$. If TargetTested does not occur, condition (3.6) holds for $\tilde{s}_{j^*,i^*}$ as well and in summary $\mathsf{kFreshWFS}(\tilde{s}_{j,i})$ holds for all sessions $\tilde{s}_{j,i}$ tested by $\mathcal{B}_4$. We bound Pr[TargetTested] below.

If the event NoRevealValidTag does not occur, then for all sessions $\tilde{s}_{j,i}$ tested by $\mathcal{B}_4$ the tag t_i output by $\mathcal{R}$ in the session $s_{i,j}$ is invalid under the real session key, which means that if Test returns the real session key, $\mathcal{B}_4$ always outputs $b = 1$. If Test returns a random key, it may happen that a tag succesfully verifies even though it is invalid under the real session key. If no such false positive occurs, $\mathcal{B}_4$ correctly outputs $b = 1$. As described in Section 3.6.1, we assume that for each tag verification the probability of a false positive is δ and $\mathcal{B}_4$ makes at most U

verifications, which implies that

$$\Pr[\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{B}_4) = 1 \mid \neg\mathsf{NoRevealValidTag} \cap \Delta] \geq \frac{1}{2} - U\delta,$$

where $\Delta \coloneqq \neg\mathsf{CorruptNotRelayed} \cap \neg\mathsf{TargetTested}$.

If NoRevealValidTag occurs, then there exists at least one session $s_{i,j}$, where $\mathcal{R}$ output a tag t_i valid under the real session key. Therefore, if TEST returns the real session key, $\mathcal{B}_4$ successfully validates t_i and correctly outputs $b = 0$ to the experiment. If TEST returns a random key, there again may be false positives. If no false positive occurs, $\mathcal{B}_4$ correctly outputs $b = 1$ to the experiment. Since $\mathcal{B}_4$ makes at most U verifications and for each verification the probability of a false positive is δ, we have

$$\Pr[\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{B}_4) = 1 \mid \mathsf{NoRevealValidTag} \cap \Delta] \geq 1 - U\delta.$$

It follows that we have

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_4) = 1 \mid \Delta] \\
&\quad \geq (\frac{1}{2} - U\delta) \cdot \Pr[\neg\mathsf{NoRevealValidTag}] + (1 - U\delta) \cdot \Pr[\mathsf{NoRevealValidTag}] \\
&\quad = (\frac{1}{2} - U\delta) \cdot (1 - \Pr[\mathsf{NoRevealValidTag}]) + (1 - U\delta) \cdot \Pr[\mathsf{NoRevealValidTag}] \\
&\quad = \frac{1}{2} - U\delta + \frac{1}{2} \cdot \Pr[\mathsf{NoRevealValidTag}]
\end{aligned}$$

In total we then have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_4) \geq \frac{1}{2} \cdot \Pr[\mathsf{NoRevealValidTag}] - \Pr[\mathsf{CorruptNotRelayed}] \\
- \Pr[\mathsf{TargetTested}] - U\delta.
\end{aligned}$$

Bounding $\Pr[\mathsf{TargetTested}]$**.** Again, let Partnered denote the set of sessions $(\tilde{s}_{j,i}, \tilde{s})$ such that $\tilde{s}_{j,i}$ is a session created by $\mathcal{B}_4$ in Step 2 and $\mathsf{Partner}(\tilde{s}_{j,i}, \tilde{s})$ holds. Then, $\tilde{s}$ must be a session created by $\mathcal{R}$ during Step 3. Additionally, since $\tilde{s}_{j,i}$ is hidden from $\mathcal{R}$, from the view of $\mathcal{R}$ the session $\tilde{s}$ has no origin session. This means that from the view of $\mathcal{R}$, it is not allowed to both test $\tilde{s}$ and corrupt the intended peer of $\tilde{s}$ as that would violate the session key freshness of $\tilde{s}$.

If CorruptNotRelayed does not occur, all parties but j^* have been corrupted in $\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U'}(\mathcal{M})$. Since we assume that $\mathcal{R}$ does not make any query, which causes it to trivially lose the experiment, for all $(\tilde{s}_{j,i}, \tilde{s}) \in \mathsf{Partnered}$, the session $\tilde{s}$ cannot have been tested by $\mathcal{R}$. If TargetTested occurs, then the session $\tilde{s}$ partnered with $\tilde{s}_{j^*,i^*}$ is the only session, for which $\mathcal{R}$ issued a TEST query. Since $\mathcal{R}$ made this query during Step 3 and all actions of $\mathcal{M}$ are independent of j^* until Step 4, $\mathcal{R}$

must have correctly guessed among the U sessions created by $\mathcal{B}_4$ which session it tests, which happens with probability $1/U$ and implies

$$\Pr[\mathsf{TargetTested}] = 1/U.$$

Summing everything up, we get the bound claimed in Theorem 2.

□

3.6.3 Discussion

Extension to Random Oracle model. Note that the argument in the proof of Theorem 2 does not rely on random oracles. Given that most highly-efficient implicitly authenticated protocols are proven secure in the random oracle model, one might ask whether it is possible to give a tightly-secure construction of Π^+ from Π in the random oracle model. For instance, one could consider computing the key confirmation message as

$$\mathsf{Conf}(k, \mathsf{pub}) \coloneqq H(k, \mathsf{pub}). \tag{3.40}$$

By modeling H as a random oracle the reduction $\mathcal{R}$ could potentially avoid committing to the key confirmation messages it simulates by just sending random strings $\widetilde{t}$ that can later be "explained" by re-programming H.

Unfortunately, we expect this approach to have an inherent tightness loss too. The reason for this is that the reduction needs to simulate the random oracle H consistently. But in order to achieve this, $\mathcal{R}$ would have to be able to distinguish a random oracle query $H(k, \mathsf{pub})$ using the "real" key (in which case it would have to return $\widetilde{t}$) from a query $H(k', \mathsf{pub})$ using an independent string k'. Since k is the session key of Π, the reduction would thus have to be able to distinguish session keys of Π from random, which should give rise to another attacker $\mathcal{B}$ on Π that proceeds as follows:

1. $\mathcal{B}$ is again a meta-reduction, which runs $\mathcal{R}$ as a subroutine, relays all queries between $\mathcal{R}$ and its security experiment, and simulates our hypothetical adversary $\mathcal{A}$ until the end of Step 3.

2. Then $\mathcal{B}$ picks an arbitrary random session $s_{i,j}$ and queries $\textsc{Test}(s_{i,j})$ to the security experiment of Π, receving back a challenge key k.

3. Now $\mathcal{B}$ issues many random oracle queries of the form $H(k_i, \mathsf{pub})$ to $\mathcal{R}$, where $k_1, ..., k_Q$ are chosen at random, but $k_\ell := k$ is defined as the challenge key k for some random index $\ell \xleftarrow{\$} \{1, \ldots, Q\}$.

4. Now either $\mathcal{R}$ is able to distinguish the query with a "real" key k from a "random" one. In this case, $\mathcal{B}$ can also distinguish the real key from a random one, by checking whether $\widetilde{t} = H(k_\ell, \mathsf{pub})$.

   Or $\mathcal{R}$ is not able to distinguish the query with a "real" key k from a "random" one. In this case, $\mathcal{R}$ will fail with probability $1 - 1/Q$, so that once again we can simulate $\mathcal{A}$ efficiently because it aborts if $\mathcal{R}$ fails.

The above proof idea is only a sketch, and we expect a rigorous proof to be significantly more complex and subtle. In this work we focused on the simpler and cleaner case of ruling out tight standard model constructions, which is also what was claimed in [CCG+19].

Stateful protocols. Note that our result not necessarily applies to stateful protocols. For example, imagine a protocol Π, where some messages depend on a randomness r that is composed as $r \coloneqq r' \parallel ctr$, where r' is a fresh source of entropy and ctr is the global number of sessions that have been started so far. For such a protocol our meta-reduction may not be able to keep the sessions it creates in Step 3 hidden from the reduction, since sessions created by $\mathcal{R}$ afterwards may reveal the existence of those sessions. However, we stress that stateful protocols are typically highly impractical as they require the synchronization of the state between multiple parties. Indeed, to the best of our knowledge, there exists no stateful AKE protocol considered practical.

3.7 Conclusion

Key confirmation is a standard technique to construct highly efficient protocols with full forward security and explicit authentication and widely used in countless protocols, such as [DvW92, Kra96, Orm98, BPR00, Kra05, Yan13a, FGSW16, CCG+19, DFW20, BMS20, GGJJ23], for example. We showed that the concrete security loss of key confirmation, which in prior work [GGJJ23] was proven for a rather restricted class of protocols and reductions, holds also for more general protocols and reductions. This suggests that this is a fundamental limitation of the widely-used key confirmation paradigm. Hence, when designing protocols with optimal concrete security, then one can either aim for maximal efficiency and use key confirmation, where avoiding a linear security loss seems difficult, or follow the approach of using digital signatures as in [BHJ+15, GJ18, DJ21, DG21, LLGW20, JKRS21], which then yields much less efficient protocols for existing constructions of digital signature schemes.

Future Research Directions. As already mentioned in Section 3.6.3, it is an open question whether our result can be extended to the random oracle model. Moreover, as the key confirmation paradigm seems promising for post-quantum cryptography, it would be interesting to explore if it can even be extended to the *quantum* random oracle model [BDF+11].

Part II

End-to-End Encrypted Backup Protocols

4 Security Analysis of the WhatsApp End-to-End Encrypted Backup Protocol

Author's contribution. The contents of this chapter are mainly based on joint work with Gareth T. Davies, Sebastian Faller, Kai Gellert, Julia Hesse, Máté Horváth, and Tibor Jager [DFG+23]. Section 4.3 is based on both [DFG+23] and joint work with Sebastian Faller, Julia Hesse, Máté Horváth, and Anja Lehmann [FHH+24], which refined the security model originally presented in [DFG+23]. As research is often an inherently collaborative task, all authors of [DFG+23] contributed equally to the description of WhatsApp's E2EE backup protocol and the development of the original security model through mutual discussions. Its refinement was developed in a similarly collaborative approach from all authors of [FHH+24].

The security analysis of the backup protocol of WhatsApp in Section 4.4.3 is a contribution from the author of this thesis. It is an extension of the original analysis from [DFG+23], which was written mostly by Sebastian Faller and Julia Hesse with minor contributions from the author of the thesis. This thesis extends the result in two ways. First, the original publication informally claims that a security analysis that only considers interactions between an honest client and a corrupt server covers all other potential combinations of corruption among a client and a server. We extend the proof to formally consider all four possible combinations of corruptions. This extension comes with several technical challenges, especially related to adaptive corruptions of either party. Second, we adjust the proof to the refined security model from [FHH+24].

4.1 Overview

The WhatsApp messenger is the most popular instant messaging app in the world with over 100 billion messages sent per day[1], providing E2EE communications [Wha21b], where no party but sender and receiver should be able to read (or modify) messages. As we have already discussed in Section 1.2, the protection of data at rest, in particular in cloud-based backups, which are enabled by default in WhatsApp, is as important as the protection of the data in transit to lay a convincing claim of E2EE for the entire WhatsApp application.

[1] See `https://x.com/wcathcart/status/1321949078381453314`

Before the end of 2021, whenever a user[2] initiated the procedure for backing up their messages (to be stored on iCloud or Google Drive), they would encrypt these messages using a key that was known to WhatsApp. While this simple approach allowed WhatsApp to return the backup encryption key to the user if the original device were to be lost, it allowed access to backed up messages beyond the control of the user. For example, by US law, federal governments could have forced WhatsApp to reveal the backup key (and the storage provider to reveal the encrypted contents) through a court order, and the previously well protected private communication suddenly becomes evidence in a lawsuit [Nov18]. More generally, the fact that the E2EE security can be circumvented by accessing the backups harbors a great potential for abuse, for instance by malfeasant governments, malicious employees, or in case of a compromise of both the storage provider's servers and WhatsApp's servers.

WhatsApp E2EE Backups. In late 2021, WhatsApp rolled out an improved protocol for protecting backups [Wha21a], with the aim to extend the E2EE security guarantees in a user-friendly way that enables users to restore their backup keys from a password in case a device is lost. The underlying protocol, which we call the WhatsApp backup protocol WBP in this work, makes use of HSMs with the core idea to outsource all cryptographic computations to the client and the HSM, while the WhatsApp "main server" essentially only relays messages (with some minor modifications) between client and HSM. The protocol is designed such that during the initialization phase[3] both client and HSM enter a secret value (a password pw for the client, and a "per-client-secret"[4] for the HSM). Furthermore, the client chooses a symmetric backup key K and the HSM receives an encrypted version of the key, without either of them learning the secret input of the counterparty. The client uses the backup key K to encrypt the backup data. If a user loses their device, they can initiate a retrieval protocol from a new client device. To this end, the new client and the WhatsApp "main server" (which, again, relays messages to the HSM) execute a protocol where the client retrieves the backup key, with the password pw used during initialization and the HSM contributing the same per-client-secret as during initialization.

In order to prevent passwords being leaked to the WhatsApp server, the WBP deploys OPAQUE [JKX18]. Further, to resist against password guessing attacks, the HSM limits the number of admissible incorrect guesses to ten [Wha21a, DLS21], after which it destroys the encrypted version of the backup key (and thus makes the backup irrecoverable).

[2] We refer to the people using a device that runs the WhatsApp client software as *users*, to the device as a *client*, and to the servers that provide the WhatsApp chat and backup service as *servers*.

[3] Note that WhatsApp refer to this phase as *registration*.

[4] It is actually a "per-backup-secret", which is determined during initialization. If a client were to re-register, a new "per-backup-secret" would be chosen.

Contributions. The WBP protocol is a widely-used real-world cryptographic protocol that addresses the fundamental problem of retrieving data from encrypted backups based on human-memorable secrets. It aims to provide strong security properties that match the E2EE-security of the messaging protocol, even against a corrupted service provider. This work presents the first rigorous security analysis of the WBP. Concretely, our results can be summarized as follows.

- We formalize the security properties expected from the WBP protocol in terms of a *password-protected key retrieval* (PPKR) scheme, where users store cryptographic keys on an untrusted server, retrievable with a password. This formalization serves as a foundation for our work, and may also support future analyses of alternative (potentially non-HSM-based) PPKR protocols and their comparison.

- We provide a *full description* of the cryptographic core of the WBP protocol. This description is based on a whitepaper published by WhatsApp [Wha21a], a public security assessment of the backup system conducted by the NCC Group [DLS21], and personal correspondence with the WBP designers [Kev23] to fill subtle but essential technical gaps left in the protocol descriptions of [Wha21a, DLS21].

- We present the *first formal security analysis* of the WBP protocol. Our analysis is conducted in the UC framework [Can01], which is simulation-based and therefore facilitates the consideration of low-entropy passwords. We formally confirm several prior statements about the security guarantees of the WBP.

- We describe how a corrupted server could get more than ten password guesses per encrypted backup, even though prior security analysis [DLS21] claimed that after ten incorrect tries the account is irremediably locked by the HSM software, and the backup data cannot be retrieved in plaintext. Concretely, we show that a corrupted server can get ten password guesses *per backup initialization*. For example, a corrupted server could suppress protocol messages to simulate a failed initialization, such that either the WhatsApp client app retries sending of initialization messages automatically, or the user re-initializes a backup manually, in order to increase the number of password guesses against the HSM.

- We give a formal analysis of the 2HashDH oblivious pseudorandom function [JKKX16] (that is used in OPAQUE) in the multi-key setting, where the domains of the two hash functions used as a building block are not assumed to be separated for different keys. For our work, this result is required since the WBP does not apply hash domain separation. Beyond that, our findings provide the basis for analyzing the *about-to-be-standardized* version

of OPAQUE [BKLW22][5] and the 2HashDH protocol currently in last call at the IRTF [DFHSW23] even under the usage of *multiple* OPRF keys.

Organization. The remainder of this chapter is structured as follows. Section 4.2 provides a full protocol description of the WBP. We then describe our model for PPKR in Section 4.3 and give our formal result in Section 4.4.

Responsible Disclosure. The research conducted for this work did not impact the entire WhatsApp system or the privacy of WhatsApp users. In particular, there was no interaction with the WhatsApp servers, HSMs, or any WhatsApp users. The protocol description was written with the help of WhatsApp employees [Kev23], and no reverse-engineering of any implemented code took place. The scenario in which a *corrupt* WhatsApp server can increase the number of password guesses against a user was never demonstrated in practice, but it was acknowledged by WhatsApp that this would indeed be possible. WhatsApp does not object to the publication.

4.1.1 Related Work

Password-protected secret sharing (PPSS) [BJSL11, JKKX16] allows a user to share a secret value among a number of servers and later retrieve it using a partial set of the servers (in the event that one or more servers become compromised or unavailable) if and only if the password used during retrieval is the same as the one used during the sharing step. This primitive has been analyzed in the UC framework [CLN12, JKKX16], and several constructions based on *oblivious pseudorandom function*s (OPRFs) exist (an overview can be found in [CHL22]).

The WhatsApp approach can be viewed as a one-out-of-one version of PPSS, where WhatsApp's HSM is the only server. This makes comparisons with work on PPSS difficult: we need to model corruption of the WhatsApp communication server (but not the HSM) and assess security in this context, something that prior models that do not split the server's role cannot capture. Nonetheless, our formalization of PPKR in the UC model takes great inspiration from existing functionalities for PPSS [JKKX16].

Beyond PPSS, there are several works that aim at bootstrapping encryption keys (or symmetric encryptions directly) from user passwords with the assistance of a server. Updatable oblivious key management [JKR19] relies on server assistance to let a user derive file-specific encryption keys from a password, while requiring strong user authentication. The distributed password-authenticated symmetric encryption scheme DPaSE [DHL22] aims for the same, while relying on the assistance of several servers but not requiring user authentication. Like the WBP,

[5]The existing formal analysis of the OPAQUE protocol [JKX18] assumes hash domain separation in 2HashDH and hence does not apply to the version of OPAQUE in the most recent Internet Draft [BKLW22].

all the above schemes rely on OPRFs to shield passwords from curious servers, but none of them aims to provide a restriction in the number of guessing attempts after the compromise of the server, which the WBP aims for.

Password-hardened encryption services [LER+18, BEL+20] let users outsource the encryption to a fully trusted frontend server. The protocols do not require OPRFs and can hence provide better throughput, at the cost of revealing the user's password to the frontend server.

The recent works on *Credential-less Secret Recovery* [Sca19, OSV23] are for a somewhat similar single-server setting as PPKR. Therein, a user stores a secret on some cloud storage and uses the additional power of a trusted execution environment (TEE) for secure recovery. In contrast to PPKR however, [OSV23] relies on a publicly accessible blockchain instead of passwords to authenticate users.

4.2 E2EE Backups in WhatsApp

In this section, we give a detailed description of the WBP. Our presentation is based on a whitepaper published by WhatsApp [Wha21a], a public security assessment of the backup system conducted by NCC Group [DLS21], and personal correspondence with WhatsApp (Meta) staff [Kev23].

We will start with a simplified explanation of the overall protocol layout in Section 4.2.1 to give a high-level overview of its main idea. Then, to prepare the detailed protocol description, we will discuss the creation of a communication channel between clients and the backup server via the WhatsApp client registration protocol in Section 4.2.2. In Section 4.2.3, we elaborate on how WhatsApp uses HSMs; in Section 4.2.4, we outline how these HSMs outsource storage to servers that are considered untrusted. In Section 4.2.5, we then provide a detailed description of the actual WBP. We conclude with Section 4.2.6, where we describe how a malicious server can increase the admissible number of password guesses in certain settings.

4.2.1 High-level Protocol Overview

There are four entities in the system: the user, a WhatsApp client running on the user's device, the WhatsApp server, and the HSM that only the WhatsApp server can communicate with. We will now focus on the latter three. In this high-level overview, we simplify by describing the WhatsApp server as merely relaying messages between the HSM and the client. In the actual WBP protocol it additionally authenticates clients and stores files encrypted by the HSM. As the encryption and outsourced storage of the data at a cloud provider is done using symmetric encryption, we focus on the initialization and retrieval of the encryption key from a password.

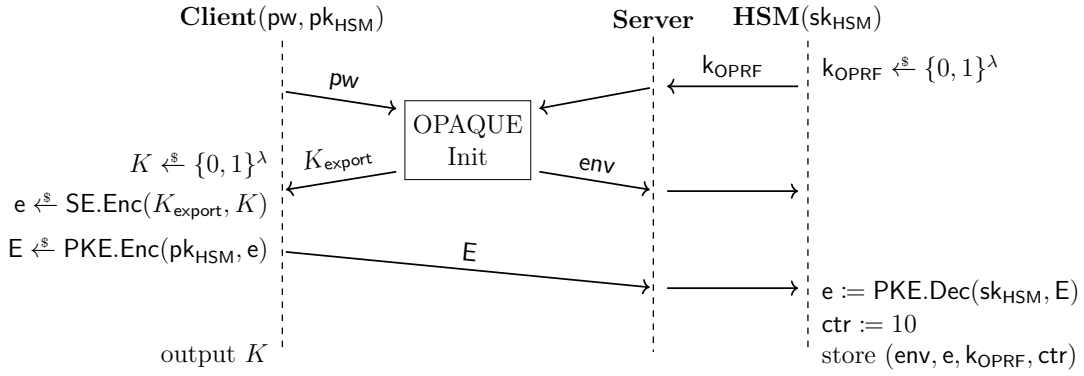

Figure 4.1: The WBP key initialization, high-level layout. The value $\mathsf{k_{OPRF}}$ is freshly chosen by the HSM for each initialization request.

Initialization. When activating WhatsApp's E2EE backups for the first time, the client chooses a backup key K to encrypt the chat history and the WBP key initialization phase is executed (see Figure 4.1). To this end, the client first runs the OPAQUE protocol with the HSM, which takes a password pw from the client and a key $\mathsf{k_{OPRF}}$ from the HSM as inputs. It then outputs a key K_{export} to the client[6] and the "envelope" env to the HSM. This envelope is encrypted under the key K_{export} (which is derived using both pw and $\mathsf{k_{OPRF}}$) and contains freshly chosen key material of the client that is used during retrieval to perform, e.g., a Diffie–Hellman key exchange, among other things.

To conclude initialization, the client encrypts the backup key K first under the symmetric key K_{export} and then under the HSM's public encryption key pk, and sends the result E to the HSM. The HSM removes the outer encryption layer and stores the encrypted backup key e, the OPAQUE envelope env, the key $\mathsf{k_{OPRF}}$, and a counter ctr initialized with 10 that tracks password guessing attempts.

Key Retrieval. If the client has lost their client device (and thus lost their backup key K), they can re-authenticate their new client device with WhatsApp (via a challenge-response protocol that takes place after re-installing the WhatsApp application), and subsequently start the retrieval part of the WBP depicted in Figure 4.2.

The first step of the retrieval phase is that client and HSM engage in the key exchange phase of the OPAQUE protocol. To this end, the client uses a value pw′ as input and the HSM contributes the values $\mathsf{k_{OPRF}}$ and env as established during the initialization phase. If the password pw′ entered by the client is equal to the password pw during initialization, the OPAQUE protocol guarantees that (1) the

[6]The option to derive this additional key was originally not part of OPAQUE [JKX18]. However, it exists in the OPAQUE Internet Draft version 03 [KLW21], which is deployed by the WBP.

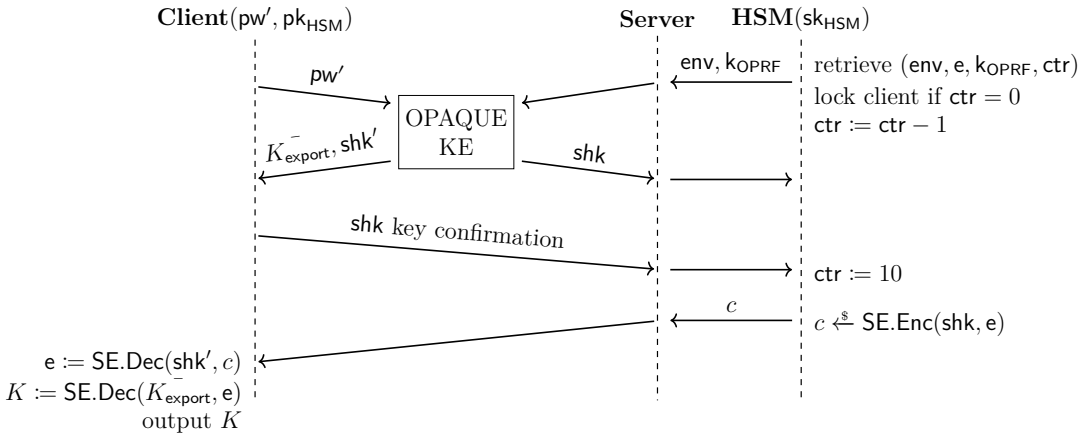

Figure 4.2: The WBP key retrieval, high-level layout.

client retrieves the former export key $\bar{K}_{\mathsf{export}} = K_{\mathsf{export}}$ and (2) both client and HSM derive the same value $\mathsf{shk}' = \mathsf{shk}$. However, if $\mathsf{pw}' \neq \mathsf{pw}$, the client will have to abort instead.

The HSM decrements its counter for attempted password guesses each time a retrieval procedure is initiated. If the client can convince the HSM that it has computed the same value $\mathsf{shk}' = \mathsf{shk}$ (via a key confirmation), the HSM will learn that the entered password pw' was indeed correct and hence reset its counter to ten and send the stored ciphertext c to the client. Lastly, the client can use the derived $\bar{K}_{\mathsf{export}}$ to decrypt the ciphertext c and retrieve their backup key K. This concludes the high-level overview of the cryptographic core of the WBP.

4.2.2 Client Registration

The WBP essentially relies on a communication channel between clients and the backup server, which is realized in an indirect way. Upon installing the WhatsApp application, the main WhatsApp server sets up a mutually authenticated channel with each new client [Wha21b]. In the WhatsApp ecosystem this is done by a server called ChatD, which is physically distinct form the WhatsApp server handling the WBP. We decided to view all WhatsApp servers as a single WhatsApp server entity, since a distinction would make the already complex protocol description and security analysis unnecessarily more complex without providing additional insight.[7]

That is, at first, a secure channel is set up between the client and WhatsApp using the Noise framework [Per]. Then WhatsApp uses SMS authentication to verify that the phone number it received via the freshly set up Noise channel belongs to

[7]For example, we want a corruption of the WhatsApp server to model a malicious WhatsApp service provider, and therefore we want to consider the entire service as corrupted in this case, without a need to distinguish between the ChatD and the backup server.

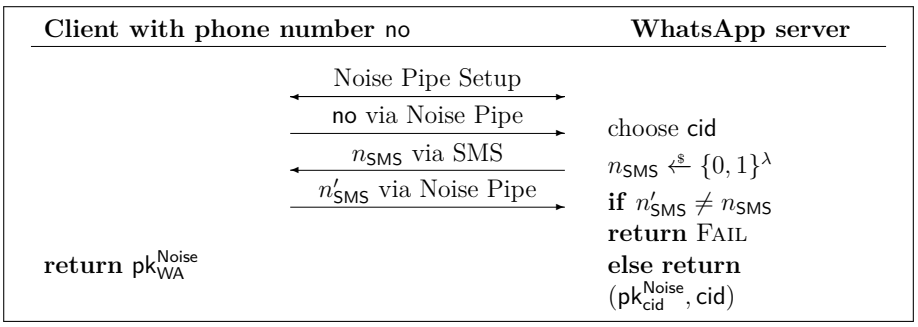

Figure 4.3: The WhatsApp client registration that is independent from the WBP. Upon conclusion, the client can be identified via a unique identifier cid, and both client and server are mutually authenticated.

the client. Upon conclusion, the client stores WhatsApp's public key $\mathsf{pk}_{\mathsf{WA}}^{\mathsf{Noise}}$ and WhatsApp stores the freshly generated client's public key $\mathsf{pk}_{\mathsf{cid}}^{\mathsf{Noise}}$ together with a unique client identifier cid. Subsequently, the WhatsApp server handles all incoming client requests via the Noise channel and also mediates the WBP messages between the client and the backup service.

4.2.3 Hardware Security Modules

The WBP deploys HSMs as a core component of their PPKR protocol. Intuitively, an HSM is a hardware device that can be programmed once with code and then "locked" in such a way that it is infeasible to change its code afterwards. After the HSMs are set up they hence serve as an incorruptible entity in the backup ecosystem. In the WBP, the HSM performs most cryptographic relevant computations on the "WhatsApp side" (with minor computations performed by a different, non-HSM WhatsApp "main server") and is responsible for coordinating the secure storage of backup keys.

Trusted Setup Ceremony. For the HSM to serve as an incorruptible entity in the system, it must be ensured that (1) its secret key material is not leaked and that (2) its code cannot be modified after setup. This is usually ensured via a process called *trusted setup ceremony* or *key ceremony*. During such a ceremony, a ceremony leader essentially unpacks new modules, sets them up with program code, and generates fresh key material for the HSM (where public keys are copied and secret keys remain secret). After setup, the HSM's "programming key" is destroyed, ensuring that it cannot be modified after the ceremony has taken place. The public key material of the HSM is hard-coded into the WhatsApp application [DLS21]. Naturally, this procedure can only be trusted if it was executed faithfully. For the remainder of this work we assume that the setup ceremony was conducted such that

- the following key material has been generated:

  - a signature key pair $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{Sig.KGen}(1^\lambda)$,
  - a public-key encryption key pair $(\mathsf{sk}_{\mathsf{Enc}}, \mathsf{pk}_{\mathsf{Enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$,
  - a static Diffie–Hellman key pair (y, Y) for $y \xleftarrow{\$} \mathbb{Z}_p$ and $Y = g^y$, where g is a generator of a known group $\mathbb{G}$ of prime order p;
- the HSM uses the secret key material to execute the WBP's computations via predefined interfaces for each protocol step;
- the HSM can only be queried by a WhatsApp server, and only via the specified interfaces. In particular, the HSM does not leak any of its secret key material.

We remark that the HSM solution deployed by WhatsApp consists of multiple HSMs, which ensures that user cannot get locked out of their backup if an HSM breaks down. All HSMs are set up such that they coordinate their state changes via the Raft consensus protocol [OO14], ensuring that each HSM behaves in the same manner towards a user [Kev23]. Without loss of generality, we treat this set of HSMs as a single HSM entity. The analysis of the HSM consensus protocol is beyond the scope of this work.

4.2.4 Secure Outsourced Storage

One might be tempted to store sensitive data along with key material in an HSM. However, storing data in the internal memory of an HSM is very expensive and limited in capacity. Thus, the internal storage of an HSM is in practice not viable to store large quantities of sensitive data for millions or billions of users. Therefore, WhatsApp uses storage fully controlled by the WhatsApp server. The HSM uses a dedicated symmetric encryption key $\mathsf{K}_{\mathsf{Enc}} \xleftarrow{\$} \{0,1\}^\lambda$ which is used for authenticated encryption, and this essentially ensures confidentiality and integrity of stored records. Whenever the HSM requests a stored record, it decrypts the record and verifies its integrity before processing it. In addition, a Merkle tree based protocol is deployed to "tie" the encrypted records together and to prevent a replay of outdated state records that were previously deleted by the HSM. Whenever the WhatsApp server provides an encrypted record to the HSM, it also has to provide a proof (using the Merkle tree) that this ciphertext is consistent with the current state of the encrypted database. The HSM verifies this against a locally stored root of the Merkle tree. If this verification fails, the HSM rejects the record.

A formal analysis of this mechanism is beyond the scope of our work. Therefore, for brevity of the protocol description and to not further complicate the security analysis, we make the simplifying assumption that the HSM has no limited storage and does not need to outsource any data. This resembles the claims made by WhatsApp [Wha21a] and the findings by the NCC Group [DLS21].

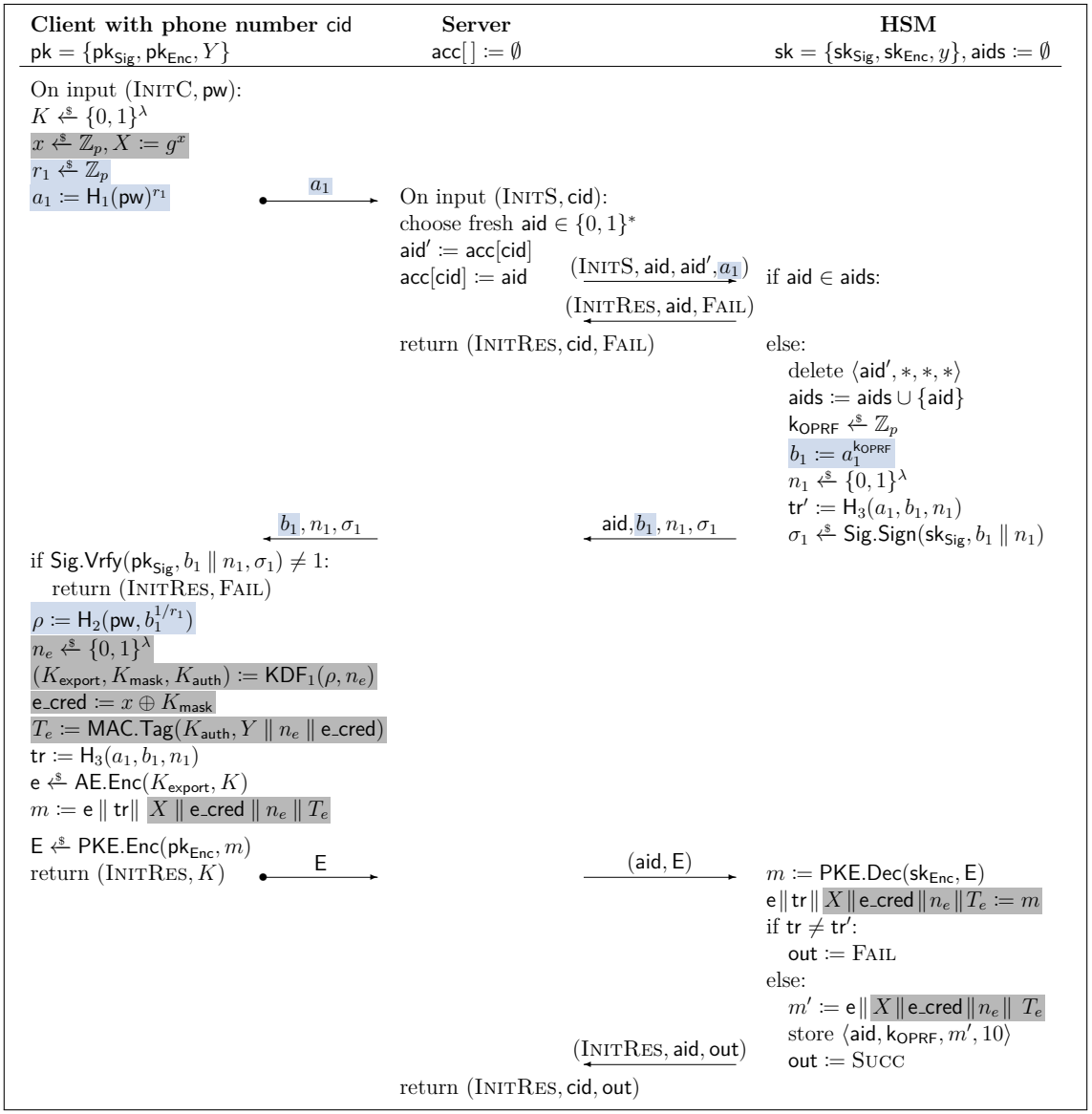

Figure 4.4: The WBP initialization. Light blue boxes indicate 2HashDH instructions of OPAQUE; dark gray boxes denote other OPAQUE instructions; non-colored parts were added by WhatsApp. $\overset{a}{\bullet\!\!\longrightarrow}$ is the cid-authenticated transmission of a.

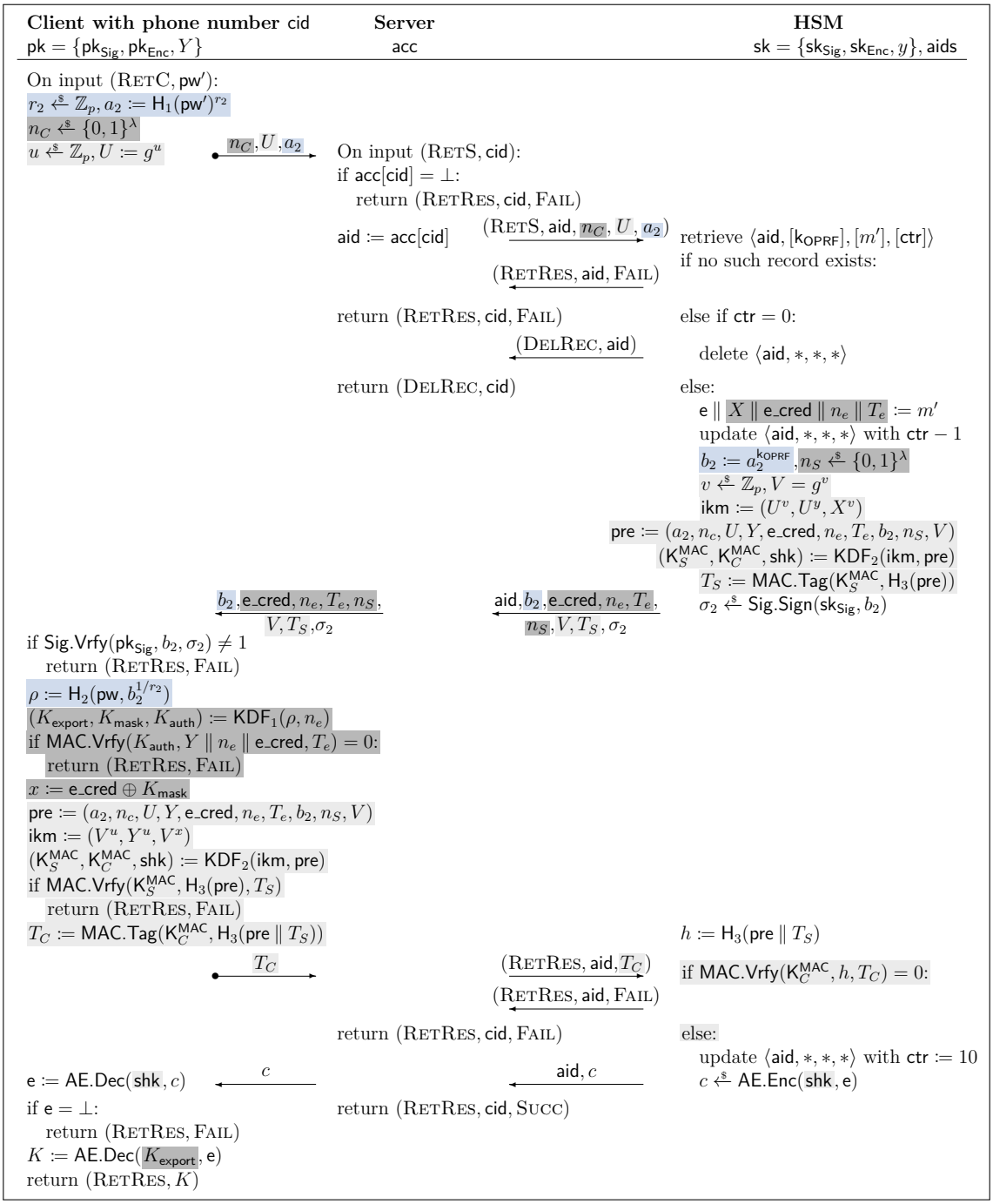

Figure 4.5: The WBP key retrieval. Light blue boxes indicate 2HashDH instructions of OPAQUE; light gray boxes mark 3DH of OPAQUE; dark gray boxes denote other OPAQUE instructions; non-colored parts were added by WhatsApp. $\overset{a}{\bullet\!\!\longrightarrow}$ is the cid-authenticated transmission of a.

4.2.5 WhatsApp Backup Protocol Description

The detailed descriptions of the WBP's key initialization and retrieval phases are depicted in Figure 4.4 and Figure 4.5, respectively. As already discussed in Section 4.2.1, the WBP builds on the OPAQUE Internet Draft v3 [KLW21], the steps of which are highlighted in the figures. For a comparison of these OPAQUE steps with [KLW21], we refer to Appendix A. Note that the figures include instructions like "On (INTERFACE, value)" or "return (INTERFACE, value)". These can bee seen as messages delivered from or to higher-level application processes, which, e.g., output a successfully established symmetric backup key to be used for backing up the actual data. We make these calls explicit since they will also appear in parts of our security model.

Participants. There are three participants in the protocol. Client refers to the WhatsApp client application of a user with unique identifier cid. The client is in possession of the HSM's public key, which is composed of a public key for a signature scheme $\mathsf{pk}_{\mathsf{Sig}}$, an encryption public key $\mathsf{pk}_{\mathsf{Enc}}$, and a static Diffie–Hellman share Y. Those keys are hard-coded into the WhatsApp client and thus authenticated.

The server is run by WhatsApp and it mostly relays messages between clients and the HSM. For this it communicates with the client via the previously established cid-authenticated channel (see Figure 4.3) and with the HSM directly through a TLS channel. The server also maintains a map $\mathsf{acc}[\cdot]$ of identifier pairs $(\mathsf{cid}, \mathsf{aid})$, which "tie" a so-called *account identifier* aid (described below) to the client identifier cid. If some cid is not contained in acc, we let $\mathsf{acc}[\mathsf{cid}] = \bot$.

Finally, the HSM is (a trusted entity that is) in possession of the secret keys $\mathsf{sk}_{\mathsf{Sig}}$, $\mathsf{sk}_{\mathsf{Enc}}$, y corresponding to the respective public keys.

Key Initialization. A user with password pw and cid initializes the backup as follows. On input of pw, the WhatsApp client app first chooses a uniformly random backup key K that can be used for encrypting the backups and which is going to be preserved via WBP. Next, it samples a Diffie–Hellman key-pair (X, x) that will be used later in the OPAQUE key exchange step. Executing the 2HashDH OPRF protocol [JKK14] with the HSM, the client samples $r_1 \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. This is then used to blind the password pw by computing $a_1 := \mathsf{H}_1(\mathsf{pw})^{r_1}$ using a hash function $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{G}$. The client sends a_1 to the server over the cid-authenticated channel.

Upon receiving a_1 from cid, the server chooses a fresh $\mathsf{aid} \in \{0,1\}^*$ that is called "account identifier" by WhatsApp[8] and checks if the client with cid has ever initiated the protocol and thus has already an aid in its array acc. If so, it

[8] We remark that this terminology is slightly misleading, as aid does not identify a client's account but is rather a "backup identifier". If the same client initializes many backups, possibly with different passwords, then each backup will be assigned a new aid and only the most recent backup is kept.

sets $\mathsf{aid}' := \mathsf{acc}[\mathsf{cid}]$ and $\mathsf{acc}[\mathsf{cid}] := \mathsf{aid}$, otherwise it sets $\mathsf{aid}' = \perp$. Finally, the server sends $\mathsf{aid}, \mathsf{aid}', a_1$ to the HSM. Observe that the HSM *never* receives any identifying information (e.g., cid) about the clients other than the value aid. That is, the HSM is *not* aware of the concept of a client cid.

Upon receiving the server's message, the HSM deletes all information associated with aid' from its storage. Then it checks whether aid has ever been used before.If it was, then the HSM aborts and outputs aid, FAIL to the server. If the HSM sees aid for the first time, it picks a random key $\mathsf{k}_{\mathsf{OPRF}} \xleftarrow{\$} \mathbb{Z}_p$ for that specific aid to be used in the 2HashDH OPRF. The HSM then uses the client's blinded password a_1 to compute $b_1 := a_1^{\mathsf{k}_{\mathsf{OPRF}}}$. Furthermore, the HSM samples a nonce $n_1 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random and signs $b_1 \| n_1$ under its secret key $\mathsf{sk}_{\mathsf{Sig}}$. Finally, it computes a transcript hash tr' of the values a_1, b_1, n_1 with a hash function $\mathsf{H}_3 \colon \{0,1\}^* \to \{0,1\}^\lambda$, and sends back aid, b_1, n_1 together with the resulting signature σ_1 to the server, which relays b_1, n_1, σ_1 to the client.

After receiving the HSM's message from the server, the client first verifies σ_1 using $\mathsf{pk}_{\mathsf{Sig}}$ and aborts if the verification fails. Otherwise, it unblinds the server's response b_1 using the randomness r_1 and derives the OPRF output $y := \mathsf{H}_2(\mathsf{pw}, b_1^{1/r_1})$ with hash function $\mathsf{H}_2 \colon \{0,1\}^* \to \{0,1\}^\lambda$. Next, further keys $K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}}$ are derived from the OPRF output y with the help of a key derivation function KDF_1. The obtained keys are used as follows. K_{mask} is used as an XOR mask to obtain $\mathsf{e\_cred}$, hiding the client's Diffie-Hellman secret x. K_{auth} is used to compute a MAC tag T_e over $Y \parallel n_e \parallel \mathsf{e\_cred}$, where n_e is a randomly sampled nonce of length λ. Finally, K_{export} is used to encrypt K to produce the envelope[9] $\mathsf{e} \xleftarrow{\$} \mathsf{AE.Enc}(K_{\mathsf{export}}, K)$. Similarly to the HSM, the client also computes a transcript hash $\mathsf{tr} := \mathsf{H}_3(a_1, b_1, n_1)$ and composes a message $m := \mathsf{e} \| \mathsf{tr} \| X \| \mathsf{e\_cred} \| n_e \| T_e$, which is then encrypted under the HSM's public key $\mathsf{E} \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{Enc}}, m)$ to hide its content from the intermediary server. The encrypted envelope E is sent to the server over the cid-authenticated channel.

Upon receiving E from a client with cid, the server looks up $\mathsf{acc}[\mathsf{cid}] := \mathsf{aid}$ and forwards aid, E to the HSM. After receiving the message, the HSM decrypts E and checks whether the received transcript hash tr matches its own view of the transcript tr'. If the transcripts do not match, it aborts sending aid, FAIL to the server that outputs cid, FAIL. In case of matching transcripts, the HSM initializes a counter ctr that aims to track the unsuccessful key retrieval attempts and stores in the secure storage the tuple $(\mathsf{aid}, \mathsf{k}_{\mathsf{OPRF}}, \mathsf{e} \parallel X \parallel \mathsf{e\_cred} \parallel n_e \parallel T_e, \mathsf{ctr})$. Finally, it informs the server about the successful completion of the initialization phase by sending aid, SUCC that outputs cid, SUCC concluding the key initialization.

Key Retrieval. The goal of the WBP retrieval phase is to enable users, who do not have access anymore to their backup key K, to retrieve it using the password they entered during key initialization. On input of pw', the WhatsApp client app first prepares its input r_2, a_2 to the 2HashDH OPRF the same way as during

[9]Note that the WBP's envelope is not equivalent to an OPAQUE envelope.

initialization, samples an ephemeral Diffie–Hellman key-pair (u, U) for the 3DH protocol, and also samples a uniformly random nonce n_C. Then n_C, U, a_2 are sent to the server over the cid-authenticated channel.

Upon receiving the client's message, the server checks the array acc and if it does not contain cid, then it aborts because no user with the identifier cid initialized any backup keys. Otherwise, it attaches the account identifier $\mathsf{aid} := \mathsf{acc}[\mathsf{cid}]$ to the client's message and sends these to the HSM.

When receiving the server's message, the HSM retrieves the record from the secure storage that is indexed by aid. If no such record is found, it returns failure to the server. Otherwise the HSM retrieves the record containing the per-client OPRF key $\mathsf{k_{OPRF}}$, the current counter value ctr, and m' that is parsed as $m' = \mathsf{e} \parallel X \parallel \mathsf{e\_cred} \parallel n_e \parallel T_e$. If $\mathsf{ctr} = 0$, then the HSM deletes the record indexed with aid from the storage and informs the server of this. If $\mathsf{ctr} > 0$, then its value is decreased by one and the stored record for aid is updated with the new ctr value. Next, as in the key initialization phase, the HSM computes $b_2 := a_2^{\mathsf{k_{OPRF}}}$ as a step of 2HashDH. After sampling a uniformly random nonce n_S, the execution of the 3DH protocol steps follows. The HSM samples an ephemeral Diffie–Hellman key pair (V, v) and computes three shared Diffie–Hellman secrets: U^v, U^y, X^v. Using these shared secrets and the preamble pre, which is essentially a concatenation of the full protocol transcript $(a_2, n_c, U, Y, \mathsf{e\_cred}, n_e, T_e, b_2, n_S, V)$, it derives the keys $\mathsf{K}_S^{\mathsf{MAC}}, \mathsf{K}_C^{\mathsf{MAC}}$, and shk from a key derivation function KDF_2. Finally, with $\mathsf{K}_S^{\mathsf{MAC}}$ it computes a MAC tag T_S over the hashed preamble, signs b_2 with $\mathsf{sk_{Sig}}$ to get signature σ_2 and sends its response composed of $\mathsf{aid}, b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2$ to the server, who removes aid from the message and forwards the rest to the client.

After receiving the HSM's response from the server, the client first verifies the signature σ_2 and aborts if the verification fails. It then again derives the keys $K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}}$ and verifies the MAC T_e that was created during the initialization. If the verification failed, it aborts. Otherwise it continues to reconstruct x by unmasking $\mathsf{e\_cred}$ with K_{mask} and then to derive the keys $\mathsf{K}_S^{\mathsf{MAC}}, \mathsf{K}_C^{\mathsf{MAC}}, \mathsf{shk}$ from the three shared Diffie–Hellman secrets V^u, Y^u, V^x and the preamble pre. Using $\mathsf{K}_S^{\mathsf{MAC}}$ it verifies the MAC T_S and aborts if this is not successful. Otherwise the client computes MAC T_C over a hash of $\mathsf{pre} \parallel T_S$ with the key $\mathsf{K}_C^{\mathsf{MAC}}$, which it then sends to the server through the cid-authenticated channel.

After attaching aid to T_C, the server forwards these values to the HSM. Since the HSM knows all values for computing the MAC tag that it has just received, it can verify the MAC. Note that with the MAC verification it essentially checks whether $\mathsf{pw} = \mathsf{pw}'$. If the MAC verification fails, it aborts, otherwise it resets the counter ctr to 10. Finally, the HSM encrypts e using shk and sends the resulting ciphertext c and aid to the server who forwards c to the client.

As the final steps of the retrieval phase, the client first decrypts c to obtain e (it aborts if the AE decryption fails) and then decrypts e using the key K_{export} to obtain the backup key K.

4.2.6 Extending the Number of Password Guesses

As we already noted in the protocol description in Section 4.2.5, the WBP only authenticates the client towards the server but not towards the HSM. The usage of the so-called "account identifier" aid aims to bridge this gap. The way it is used ensures that the HSM always associates every retrieval request from the same cid with the same unique aid that was assigned for this cid during its last successful key initialization request. Furthermore, the HSM only keeps records of the last key initialization of a user under the aid that was assigned to the corresponding cid during this last initialization. Recall that in order to limit the number of password guesses against some account, each password guess has to be associated with the targeted account. It turns out that this cannot be guaranteed in case of the WBP when the server is malicious. The reason for this is that the server is in charge of assigning aids for cids, and neither the client nor the HSM can check this because the former never sees their assigned aid, and the latter never learns the cid of clients. This allows the server to increase the number of password guesses in certain cases.

We demonstrate the attack with an example. Let us assume that some client with identity cid has already initialized a key and the HSM stored the corresponding record under aid. Now if the same client runs key initialization again with the same password,[10] the server is assumed to instruct the HSM to delete the previous record by setting $\mathsf{aid}' := \mathsf{aid}$. However, the execution of this step completely depends on the server acting honestly. A malicious server might however proceed as if it has never seen cid before and make the HSM store q_I records for cid, if the client with cid runs the key initialization q_I times. If the client used the same password pw all q_I times, the malicious server will have $10q_I$ password guessing opportunities, since it knows all the q_I aids that are associated with cid's records.

Mitigating the Attack. If the transcripts tr' and tr contained information about the client identity cid, in a way that both the client and the HSM can verify this, then they would be able to notice if the server is dishonest about the client identity. However, note that this countermeasure is very difficult to deploy retroactively, since any changes in the programming of the HSM would require the setup ceremony to be performed again.

Resetting ctr without the correct password. Recall that the HSM determines whether a retrieval was successful solely by verifying the MAC T_C. This implies that the ability to compute a valid T_C in fact does not prove knowledge of the correct password. Instead it proves knowledge of the DH secret x corresponding to the DH share X stored by the HSM, which is needed to compute $\mathsf{K}_C^{\mathsf{MAC}}$ and in turn a valid T_C. A notable consequence is therefore that any party that knows x

[10] WhatsApp is for mobile devices, connection loss may happen leading to a failure. After an unsuccessful attempt, the user would most probably re-run initialization, likely with the same password.

can always reset the counter ctr to 10 even if it runs the retrieval with a wrong password. The practical impact of this, however, is rather small as in this way a corrupt party can only reset the ctr of a file that it initialized itself. An adversary cannot exploit this to reset the ctr of any *other* party, even when a maliciously acting server executes the attack described above and reroutes the retrieval to a different aid.

4.3 Password-Protected Key Retrieval

In this section we give a formal definition of password-protected key retrieval (PPKR) in terms of an ideal functionality.

The Cryptographic Abstraction of the WBP. We introduce the concept of a *password-protected key retrieval* (PPKR) protocol, which is a 2-party protocol executed by a client and a server. Note that in the WBP protocol, this client is the user's WhatsApp client and this server is the combination of the WhatsApp server and the HSM. A PPKR protocol consists of two phases: (1) an initialization phase, where the client generates a symmetric encryption key and, using a password, securely stores it with a server, and (2) a retrieval phase, where the client can retrieve their symmetric encryption key using a password.

The server may neither learn any information about the client's password, nor their key from these interactions, but only whether a retrieval was successful. To provide a high-level intuition, we depict the input-output behavior of the PPKR functionality in Figure 4.6. Besides this, we demand several properties from a PPKR scheme that seem relevant for such a primitive in practice. These include protection against online and offline dictionary attacks, and that it provides cryptographically strong encryption keys.

Remark 3. We note here that alternative modelings are possible. For example, one could case-tailor the definition to the WhatsApp setting and formulate a variant of PPKR with three parties: the client, the server, and the HSM. However, our notion with only two parties is more versatile: it can be used to analyze protocols where the server relies on an HSM, as well as protocols where the server acts on its own, or relies on arbitrarily many other entities, such as a cloud provider, offline storage, etc. The reason why this is possible is that usage of such "helpers" is well integrated into the UC framework [Can01] through the notion of so-called *hybrid functionalities*, which can be modularly "plugged" into protocol descriptions without "spilling" into the definition of the underlying primitive.

4.3.1 Modeling Server (and HSM) Corruption

The central entity in our system is the server S, which provides the password-protected key retrieval service to its clients, and which we model to be corruptible in several ways. As already mentioned above, the HSM does not appear as an

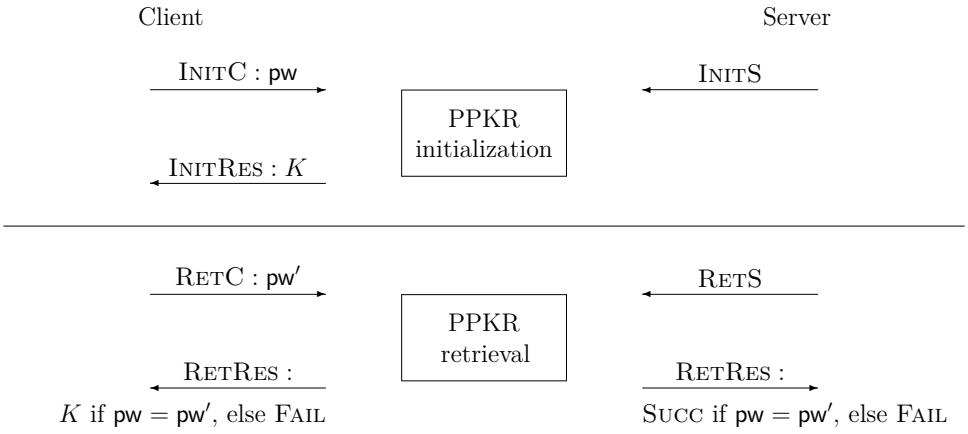

Figure 4.6: Schematic overview of password-protected key retrieval with initialization on top and retrieval at the bottom, already using the interface names of our functionality $\mathcal{F}_{\mathsf{PPKR}}$. In both phases, the server does not provide any particular input but still has to participate in the protocol for the client to successfully initialize and retrieve a key K, which is modeled in the ideal functionality by letting it provide the InitS and RetS messages.

explicit entity in a PPKR protocol, but is rather considered an artifact on how the server's code is deployed. Let us discuss the different corruption scenarios that we consider in this setting.

Server Corrupt: This corruption status gives a corrupt server some, rather benign, attack capabilities, e.g. as discussed in Section 4.2.6, but still maintains most security guarantees. This scenario models that the server essentially mainly acts as a connecting interface between clients and the HSM (or any other "helper" utilized by the server), which remains perfectly secure and leaks no information. Applying this scenario to the WBP, this in particular means that all cryptographic key material and client records remain protected during this corruption thanks to being executed in the HSM.

File Leakage: We additionally model that the client files, containing the information to verify the recovery password and retrieve the client's key, can be leaked to the adversary. In the context of an HSM-supported setup, this means that the adaptively stored and maintained information by the HSM can get compromised. The file leakage can happen repeatedly, with the adversary obtaining snapshots of all the stored client files. This does neither imply that the server nor HSM are corrupt though.

We believe this realistically models that the core key material of the HSM enjoys particularly strong protection, whereas the HSM-protected database of possibly millions of client records is less secure and can be vulnerable to leakage attacks.

Server (and HSM) Fully Corrupt: We also want to capture the (loss of) security when the server gets fully corrupted, meaning that all parts of the server—including the HSM (or any other helper)—are under the control of the adversary. This implies the previous two corruption settings, i.e., normal server corruption and *continuous* compromise of all files, and also goes beyond as we now assume that the adversary is in full control of the HSM. That is, the core key material is no longer trusted, and the adversary can arbitrarily deviate from the original HSM protocol.

Expected Security Properties. Before formalizing an ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$, we first describe the security properties we intuitively expect from a PPKR scheme and how they degrade under the discussed corruption scenarios.

The key K that is generated in a PPKR scheme should be usable without restrictions in any other application, thus we require that K is indistinguishable from random. Note that in particular this implies that neither the server nor the adversary can influence the generation of K in any way. Moreover, we expect K to remain secret from everyone but the client that computed it throughout the lifetime of the PPKR protocol. Hence, $\mathcal{F}_{\mathsf{PPKR}}$ should leak no information on the key, and any output given by $\mathcal{F}_{\mathsf{PPKR}}$ to the server or the adversary must be independent from K. These two properties should even hold in any corruption scenario, unless the adversary correctly guesses the client's password.

In order to limit the ability to guess passwords via dictionary attacks, we expect $\mathcal{F}_{\mathsf{PPKR}}$ to (1) not leak any information about a password used by some client to any other party beyond whether a password used in a retrieval is the same as in the most recent initialization by that client, and (2) allow only a small number of failed retrieval attempts before K is deleted, although there are exceptions to both properties under full corruption.

Regarding (1), a fully corrupt server has extended (although still limited) online[11] password guessing capabilities against an honest cid in the following sense. A fully corrupt S is not limited to running a retrieval with the currently stored backup file of cid. Instead, it can use currently stored or outdated backup files of any client, or even arbitrary data. This means that S not necessarily learns whether the password used by cid in the retrieval is equal to the one from the most recent initialization by cid, but can compare it to the password from almost any initialization[12] or even arbitrary passwords. This also implies that S can choose which key cid obtains if the passwords match—be it the currently stored or outdated key of any cilent, or an arbitrary one. Note that in any case, the server is still limited to a single password guess per retrieval.

The exception regarding (2) is that the limit on retrieval attempts cannot be

[11] We refer to attacks that require interaction with at least one other party as *online*. Accordingly, *offline* attacks require no interaction with other parties.

[12] This excludes initializations, for which the file was never leaked and got deleted (through exhaustion of retrieval attempts or re-initialization) before the server became fully corrupt.

guaranteed anymore if the server is fully corrupt as we cannot guarantee that it actually deletes K. We emphasize that file leakage in contrast has no impact on the number of possible failed retrieval attempts as this is only a passive one-time access to the files and gives no active ability to tamper with them or the counter. However, leaking file gives the adversary an unlimited number of offline password guessing attempts. Nevertheless, we expect that a PPKR scheme provides resistance against pre-computation attacks [JKX18] for leaked files, meaning that the adversary cannot pre-compute a table of possible passwords and immediately learn the user's password upon file leakage.

We further expect that clients are authenticated towards the server[13] and cannot be impersonated by other clients or the adversary. However, note that a corrupt server might be able to skip client authentication[14] and execute the protocol on behalf of any client by itself, i.e., without interacting with any client. Finally, we expect that the server is authenticated towards clients and cannot be impersonated by the adversary.

We summarize the list of expected security properties below.

- **Pseudorandomness of K**: Honest clients compute pseudorandom keys K, even if the server is fully corrupt.
- **Secrecy of K**: Initialized and retrieved keys of any honest client remain hidden from even a fully corrupt server as long as the server does not correctly guess the honest client's password.
- **Uniqueness of K**: If the server is honest, initialization of a key K by user cid deletes any key that cid previously initialized.
- **Oblivious passwords**: The initialization phase does not leak any information about the password to even a fully corrupt server.
- **No online dictionary attacks**: The retrieval phase leaks at most one bit of information (1) about the password used by the client to even a fully corrupt server, and (2) about the initialized password to even a malicious client.
- **Limited number of retrieval attempts with wrong passwords**: Let K denote the key initialized by an honest client cid, and assume that cid later runs the retrieval phase 10 times consecutively with a wrong password.

[13] We leave the concrete means of authentication to the application. In the case of the WBP, SMS-based authentication is used, creating a one-to-one correspondence between cid and phone numbers of WhatsApp users. Other authentication methods such as biometrics (where cid would correspond to, e.g., a fingerprint) or even device-bound strong authentication using signatures are possible as well.

[14] We opted for a general treatment here, i.e., allowing client impersonation by the server. In fact, we could strengthen this (see Section 4.3.3 for more details) but this depends on which mechanisms on the server side are corruptible.

Then, unless the server is fully corrupt, it erases all K-dependent information, i.e., K cannot be retrieved anymore. This must hold even if the client becomes maliciously corrupted after initializing K.

- **Limited number of offline guesses**: The above guarantee extends to corrupt servers if there was no file leakage, i.e., after 10 wrong password guesses to retrieve any honestly initialized key K, K is deleted and cannot be retrieved by anyone anymore if no file containing K was leaked.[15]
- **Resistance to pre-computation attacks**: Secrecy of K is maintained even for leaked files.
- **Client authentication**: As long as the server is honest, only the client who initialized K can attempt to retrieve K. This must hold even if the password used during initialization becomes publicly known.
- **Server authentication**: There is only one server in the system and it cannot be impersonated by the adversary, unless the server gets corrupted.
- **Key authenticity**: Let K be the key initialized by any client cid. Then, if cid retrieves with the correct password, it obtains K as long as the server is honest.

We give an overview and comparison of the main security properties in the different corruption settings in Table 4.1 below. For brevity, we refer to security that is maintained when the server S is corrupt, but no files got leaked as Lev-1 security. Lev-2 security covers the optimal guarantees up to joint server corruption and file leakage, considering both attack capabilities combined. Finally, Lev-3 security refers to schemes that maintain the optimal security guarantees up to full server corruption.

4.3.2 A PPKR Functionality

In Figures 4.7 to 4.9 we describe the ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$ for password-protected key retrieval. On a high level, $\mathcal{F}_{\mathsf{PPKR}}$ implements a password protected lookup table that contains clients' keys. When some client executes the initialization phase, an entry for that client is created in the lookup table or updated if an entry already existed. By executing the retrieval phase, clients can access their entry in the table and recover their key, but only if they pass password authentication. If they fail password authentication 10 times in a row, $\mathcal{F}_{\mathsf{PPKR}}$ deletes the key by erasing the corresponding table entry of that user. Note that while $\mathcal{F}_{\mathsf{PPKR}}$

[15] Note that the phrasing "any initialized" here reflects that the adversary can extend the number of admissible password guesses, as described in Section 4.2.6. This is necessary to model the security achieved by WhatsApp's protocol. We will discuss in Section 4.3.3 how the functionality can be strengthened.

Table 4.1: Overview of expected security properties of $\mathcal{F}_{\mathsf{PPKR}}$ under different corruption settings. We refer to the following level: Lev-1 = S is corrupt, Lev-2 = S is corrupt and files are leaked (both attacks combined), Lev-3 = fully corrupt S.

Property	Honest S	Lev-1	Lev-2	Lev-3
Pseudorandom key K	✓	✓	✓	✓
Secrecy of K	✓	✓	✓	✓
Uniqueness of K	✓	✗	✗	✗
Oblivious passwords	✓	✓	✓	✓
No online dictionary attacks	✓	✓	✓	✓
Upper limit on incorrect recoveries	✓	✓	✓	✗
Limited offline attacks	✓	✓	✗	✗
No precomputation for offline attacks	n.a.	n.a.	✓	✓
Client authentication	✓	✗	✗	✗
Server authentication	✓	✗	✗	✗
Key authenticity	✓	✗	✗	✗

maintains the table entries using client *identifiers*, $\mathcal{F}_{\mathsf{PPKR}}$ does not enforce the initialization and retrieval processes to run on the same physical client machine. In our model, we understand the client's party identifier as the identity under which the client device can authenticate. This way, if multiple devices can authenticate under the same identity, InitC and RetC can be called from different machines.

Next, we will explain the interfaces and record keeping of $\mathcal{F}_{\mathsf{PPKR}}$. In Figures Figures 4.7 to 4.9 we labeled all instructions for easy referencing. $\mathcal{F}_{\mathsf{PPKR}}$ interacts with arbitrary clients and a single server S, where S is encoded in the session identifier sid (**server authentication**). Note that for brevity we omit sid from all inputs and outputs of $\mathcal{F}_{\mathsf{PPKR}}$. The functionality internally maintains different types of records to keep track of ongoing and finished initialization and retrieval phases. If $\mathcal{F}_{\mathsf{PPKR}}$ ever tries to retrieve a record that does not exist, it ignores the query causing this.

Initialization Phase. Whenever a client cid starts a new initialization, it calls the InitC interface with the password pw the user has chosen and a subsession identifier ssid. $\mathcal{F}_{\mathsf{PPKR}}$ then (IC.1) generates a fresh key $K \xleftarrow{\$} \{0,1\}^{\lambda}$ (ensuring **pseudorandomness** of K) for cid and records that cid has started a new initialization by creating a record $\langle \text{InitC}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K, \bot, \bot \rangle$ (IC.2). As the functionality enforces no order in which the InitC and InitS interface are called, it may happen that the InitS interface was already queried previously, in which case $\mathcal{F}_{\mathsf{PPKR}}$ updates the record $\langle \text{InitC}, \mathsf{ssid}, \mathsf{cid}, \bot, \bot, *, \mathtt{srv0k} \rangle$ created in the InitS interface with the values pw and K (IC.2). Finally, the functionality informs the adversary $\mathcal{A}$ that the client cid has started a new initialization (IC.3).

If the server agrees to participate in the initialization with cid in the subses-

$\mathcal{F}_{\mathsf{PPKR}}$ is parameterized with a security parameter λ. $\mathcal{F}_{\mathsf{PPKR}}$ talks to a server S that is encoded in sid, arbitrary clients cid, and the adversary $\mathcal{A}$. S can have corruption state HONEST, CORRUPT, or FULLYCORRUPT. $\mathcal{F}_{\mathsf{PPKR}}$ further allows file leakage via the LEAKFILE corruption command. For brevity, we omit session identifier sid from all inputs, outputs, and records. If $\mathcal{F}_{\mathsf{PPKR}}$ tries to retrieve a record that does not exist, it ignores the incoming message. $\mathcal{F}_{\mathsf{PPKR}}$ ignores repeated inputs with the same ssid and in that case activates $\mathcal{A}$.

Initialization phase

On input $(\text{INITC}, \mathsf{ssid}, \mathsf{pw})$ from cid:

- IC.1 Choose $K \xleftarrow{\$} \{0,1\}^\lambda$
- IC.2 If a record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, \bot, \bot, *, \mathtt{srv0k} \rangle$ exists, overwrite $(\bot, \bot)$ in it with (pw, K). Otherwise record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K, \bot, \bot \rangle$
- IC.3 Send $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{A}$

On input $(\text{INITS}, \mathsf{ssid}, \mathsf{cid}, \boxed{\mathsf{cid}'})$ from S:

- IS.1 If a record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, *, *, \bot, \bot \rangle$ exists, overwrite the last $\bot$ with $\mathtt{srv0k}$. Otherwise record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, \bot, \bot, \bot, \mathtt{srv0k} \rangle$
- IS.2 If S is not HONEST, in the record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, *, *, \bot, \mathtt{srv0k} \rangle$ overwrite $\bot$ with cid'
- IS.3 Send $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{A}$

On input $(\text{COMPLETEINITC}, \mathsf{ssid}, b)$ from $\mathcal{A}$:

- CIC.1 Retrieve $\langle \text{INIT}, \mathsf{ssid}, [\mathsf{cid}], *, [K], *, \mathtt{srv0k} \rangle$. Ignore the query if $K = \bot$
- CIC.2 If $b = 1$, output $(\text{INITRES}, \mathsf{ssid}, K)$ to cid
- CIC.3 If $b = 0$, output $(\text{INITRES}, \mathsf{ssid}, \text{FAIL})$ to cid

On input $(\text{COMPLETEINITS}, \mathsf{ssid}, b)$ from $\mathcal{A}$:

- CIS.1 Retrieve $\langle \text{INIT}, \mathsf{ssid}, [\mathsf{cid}], [\mathsf{pw}], [K], [\mathsf{cid}'], \mathtt{srv0k} \rangle$. Ignore the query if $K = \bot$. If $\mathsf{cid}' \neq \bot$, set $\mathsf{cid} := \mathsf{cid}'$
- CIS.2 If $b = 0$, output $(\text{INITRES}, \mathsf{ssid}, \mathsf{cid}, \text{FAIL})$ to S. Else continue.
- CIS.3 Set $\mathsf{ctr} := \infty$ if S is FULLYCORRUPT, and $\mathsf{ctr} := 10$ otherwise. Store $\langle \text{FILE}, \mathsf{cid}, \mathsf{pw}, K, \mathsf{ctr}, \boxed{\text{HONEST}} \rangle$ if cid is honest and $\langle \text{FILE}, \mathsf{cid}, \mathsf{pw}, K, \mathsf{ctr}, \boxed{\text{MALICIOUS}} \rangle$ otherwise, overwriting any existing record $\langle \text{FILE}, \mathsf{cid}, *, *, *, * \rangle$
- CIS.4 If S is FULLYCORRUPT, record $\langle \text{LEAKED}, \mathsf{cid}, \mathsf{pw}, K, i+1 \rangle$, where $i \in \mathbb{N}$ is the largest number, such that a record $\langle \text{LEAKED}, \mathsf{cid}, *, *, i \rangle$ exists, or $i = 1$ if no such record exists.
- CIS.5 Send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to S

Figure 4.7: Ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$ for password-protected key retrieval, initialization interfaces. Code in solid boxes and dashed boxes can be dropped to strengthen $\mathcal{F}_{\mathsf{PPKR}}$ (see Section 4.3.3).

Retrieval Phase

On input $(\text{RETC}, \mathsf{ssid}, \mathsf{pw}')$ from cid:

- RC.1 If a record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \bot, *, [\mathsf{pw}], *, \mathtt{srv0k}\rangle$ exists, overwrite $\bot$ with pw' and set $match := (\mathsf{pw} \stackrel{?}{=} \mathsf{pw}')$. Otherwise record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}', \bot, \bot, \bot, \bot\rangle$ and set $match := \bot$
- RC.2 Send $(\text{RETC}, \mathsf{ssid}, \mathsf{cid}, match)$ to $\mathcal{A}$

On input $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \boxed{\mathsf{cid}'}, \mathsf{pw}^*, K^*, i)$ from S:

- RS.1 If a record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, [\mathsf{pw}'], \bot, \bot, \bot, \bot\rangle$ exists, overwrite the last entry with $\mathtt{srv0k}$. Otherwise set $\mathsf{pw}' := \bot$ and record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}', \bot, \bot, \bot, \mathtt{srv0k}\rangle$
- RS.2 If S is not HONEST, in the record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}', \bot, *, *, *\rangle$ overwrite $\bot$ with cid' and set $\mathsf{cid} := \mathsf{cid}'$
- RS.3 If S is FULLYCORRUPT and $\mathsf{pw}^* \neq \bot$: set $\widehat{\mathsf{pw}} := \mathsf{pw}^*$, $\widehat{K} := K^*$
- RS.4 If S is FULLYCORRUPT, $\mathsf{pw}^* = \bot$, retrieve the record $\langle \text{LEAKED}, \mathsf{cid}, [\mathsf{pw}], [K], i\rangle$ and set $\mathsf{ctr} := \infty$. Otherwise, retrieve the record $\langle \text{FILE}, \mathsf{cid}, [\mathsf{pw}], [K], [\mathsf{ctr}], *\rangle$. Set $\widehat{\mathsf{pw}}, \widehat{K}$ as follows:
  (1) If no such record exists, set $\widehat{\mathsf{pw}} := \bot, \widehat{K} :=$ FAIL.
  (2) If $\mathsf{ctr} = 0$, set $\widehat{\mathsf{pw}} := \bot, \widehat{K} :=$ DELREC and delete $\langle \text{FILE}, \mathsf{cid}, *, *, *, *\rangle$.
  (3) Else, set $\widehat{\mathsf{pw}} := \mathsf{pw}, \widehat{K} := K$ and overwrite ctr with $\mathsf{ctr} - 1$ in the FILE record.
- RS.5 In the record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}', *, \bot, \bot, *\rangle$ overwrite $(\bot, \bot)$ with $(\widehat{\mathsf{pw}}, \widehat{K})$
- RS.6 If $\mathsf{pw}' = \bot$, set $match := \bot$, else set $match := (\widehat{\mathsf{pw}} \stackrel{?}{=} \mathsf{pw}')$
- RS.7 Send $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, match)$ to $\mathcal{A}$.

On input $(\text{COMPLETERETC}, \mathsf{ssid}, b)$ from $\mathcal{A}$:

- CRC.1 Retrieve $\langle \text{RET}, \mathsf{ssid}, [\mathsf{cid}], [\mathsf{pw}'], *, [\mathsf{pw}], [K], \mathtt{srv0k}\rangle$. Ignore the query if $\mathsf{pw}' = \bot$.
- CRC.2 Determine the output K' as follows:
  (1) If $K \in \{\text{FAIL}, \text{DELREC}\}$, set $K' := K$
  (2) If $\mathsf{pw} = \mathsf{pw}'$ and $b = 1$, set $K' := K$
  (3) In all other cases, set $K' :=$ FAIL
- CRC.3 Send $(\text{RETRES}, \mathsf{ssid}, K')$ to cid.

On input $(\text{COMPLETERETS}, \mathsf{ssid}, b)$ from $\mathcal{A}$:

- CRS.1 Retrieve $\langle \text{RET}, \mathsf{ssid}, [\mathsf{cid}], [\mathsf{pw}'], [\mathsf{cid}'], [\mathsf{pw}], [K], \mathtt{srv0k}\rangle$. Ignore the query if $\mathsf{pw}' = \bot$. If $\mathsf{cid}' \neq \bot$, set $\mathsf{cid} := \mathsf{cid}'$
- CRS.2 Determine the output $result$ as follows:
  (1) If $K \in \{\text{FAIL}, \text{DELREC}\}$, set $result := K$.
  (2) If $b = 1$ and either $\mathsf{pw} = \mathsf{pw}'$ or a record $\langle \text{FILE}, \mathsf{cid}, *, *, *, \text{MALICIOUS}\rangle$ exists, set $result :=$ SUCC.
  (3) In all other cases, set $result :=$ FAIL.
- CRS.3 If $result =$ SUCC and there exists a record $\langle \text{FILE}, \mathsf{cid}, \mathsf{pw}, K, [\mathsf{ctr}], *\rangle$, overwrite ctr with 10.
- CRS.4 Send $(\text{RETRES}, \mathsf{ssid}, result)$ to S.

Figure 4.8: Ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$, retrieval interfaces. Code in solid boxes and dashed boxes can be dropped to strengthen $\mathcal{F}_{\mathsf{PPKR}}$ (see Section 4.3.3).

Corruption interfaces

On corruption command (LeakFile) from $\mathcal{A}$:

- LF.1 Create an empty list L.
- LF.2 For every record $\langle \text{File}, [\mathsf{cid}], [\mathsf{pw}], [K], [\mathsf{ctr}], * \rangle$:
  (1) Append $(\mathsf{cid}, \mathsf{ctr})$ to L.
  (2) If no record $\langle \text{leaked}, \mathsf{cid}, *, *, * \rangle$ exists, add a record $\langle \text{leaked}, \mathsf{cid}, \mathsf{pw}, K, 1 \rangle$.
  (3) Otherwise, add a record $\langle \text{leaked}, \mathsf{cid}, \mathsf{pw}, K, i+1 \rangle$, where $i \in \mathbb{N}$ is the largest number, such that a record $\langle \text{leaked}, \mathsf{cid}, *, *, i \rangle$ exists.
- LF.3 Output L to $\mathcal{A}$.

On corruption command (Corrupt, P) from $\mathcal{A}$:

- C.1 Mark P as corrupt.

On corruption command (FullyCorrupt, S) from $\mathcal{A}$:

- FC.1 Mark S as FullyCorrupt and run LS.1-LS.2
- FC.2 In every record $\langle \text{File}, *, *, *, [\mathsf{ctr}], * \rangle$ overwrite ctr with ∞.
- FC.3 Output L to $\mathcal{A}$

Attack interfaces

On input (MaliciousInit, $\mathsf{cid}, \mathsf{pw}^*, K^*$) from $\mathcal{A}$:

- MI.1 If S is honest, ignore this input.
- MI.2 Record $\langle \text{File}, \mathsf{cid}, \mathsf{pw}^*, K^*, 10, \boxed{\text{Malicious}} \rangle$, overwriting any existing record $\langle \text{File}, \mathsf{cid}, *, *, *, * \rangle$

On input (MaliciousRet, $\mathsf{cid}, \mathsf{pw}^*, b$) from $\mathcal{A}$:

- MR.1 If S is honest, ignore this input.
- MR.2 If no record $\langle \text{File}, \mathsf{cid}, [\mathsf{pw}], [K], [\mathsf{ctr}], [st] \rangle$ exists, output Fail to $\mathcal{A}$.
- MR.3 If $\mathsf{ctr} = 0$, delete record $\langle \text{File}, \mathsf{cid}, \mathsf{pw}, K, \mathsf{ctr}, * \rangle$ and output (DelRec, cid) to $\mathcal{A}$
- MR.4 If $\mathsf{pw}^* = \mathsf{pw}$ or if $st = \text{Malicious}$ and $b = 1$, overwrite ctr in the record with 10 and output K to $\mathcal{A}$. Otherwise, overwrite ctr with $\mathsf{ctr} - 1$ and output Fail to $\mathcal{A}$

On input (OfflineAttack, $\mathsf{cid}, \mathsf{pw}^*, i$) from $\mathcal{A}$:

- OA.1 If a record $\langle \text{leaked}, \mathsf{cid}, \mathsf{pw}^*, [K], i \rangle$ exists, output K to $\mathcal{A}$. Otherwise, output Fail to $\mathcal{A}$.

Figure 4.9: Ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$, corruption and attack interfaces. The boxed code can be dropped to strengthen $\mathcal{F}_{\mathsf{PPKR}}$ (see Section 4.3.3).

sion ssid, it calls the InitS interface, which takes as input two (not necessarily distinct) client identities cid and cid′. The additional identity cid′ is only effective if the server is not honest (IS.2), and reflects that a malicious server can simply ignore client authentication and claim a different identity has authenticated to him. Similar to the InitC interface, the functionality either creates a new record or updates an existing one depending on the order in which the two interfaces are called. Here it stores the identifier `srvOk`, which simply indicates that the server has agreed to participate in this initialization, and potentially the additional identity cid′ (IS.1 and IS.2). Finally, $\mathcal{F}_{\mathsf{PPKR}}$ informs the adversary that S agreed to the initialization with cid.

After both parties agreed to participate in the initialization, the adversary may let $\mathcal{F}_{\mathsf{PPKR}}$ compute the output of the initialization phase for the client with the interface CompleteInitC and for the server with CompleteInitS. Both interfaces verify that the respective other party gave its InitC or InitS input (cf. CIC.1 and CIS.1) by ensuring that the session record contains a key K (stored in IC.2) and the identifier `srvOk` (stored in IS.1). Again, the functionality does not enforce an order in which the parties receive their output and leaves this decision to the adversary $\mathcal{A}$. The adversary's choices are as follows:

- CompleteInitC with the parameter $b = 1$ outputs K from the session record to the client (CIC.2)
- CompleteInitC with the parameter $b = 0$ outputs Fail to the client (CIC.3)
- CompleteInitS with the parameter $b = 1$ outputs Succ to the server (CIS.5)
- CompleteInitS with the parameter $b = 0$ outputs Fail to the server (CIS.2)

Further, if it does not output Fail, the CompleteInitS interface ensures that $\mathcal{F}_{\mathsf{PPKR}}$ installs a password-protected backup key file for cid that contains K (CIS.3). Note that here the additional identity cid′ from the InitS interface becomes effective. If it is present, the backup key file is created for cid′ instead of cid, which reflects that a malicious server can claim that any cid′ completed the initialization (CIS.1). Moreover, the file contains a counter ctr initialized to 10 that indicates the remaining retrieval attempts before the file gets deleted. If the server is fully corrupt, ctr is instead initialized to ∞, indicating that a fully corrupt server may never delete files. The last entry of the installed file marks whether the initializing client is honest or corrupt, which will later be necessary in the retrieval to reflect that corrupt parties can always reset their counter even when retrieving with a wrong password.

Finally, since a fully corrupt server implies a continuous compromise of all files, $\mathcal{F}_{\mathsf{PPKR}}$ creates essentially a copy of the installed file in a leaked record, which is utilized by $\mathcal{F}_{\mathsf{PPKR}}$ to enable offline password guesses for any leaked backup file (CIS.5).

This concludes the description of $\mathcal{F}_{\mathsf{PPKR}}$'s initialization phase. The absence of any K- or pw-dependent information in the outputs towards the server (CIS.2 ,

CIS.5) and the adversary (IC.3, IS.3) ensures **secrecy of** K and **password obliviousness** during initialization. Since $\mathcal{F}_{\mathsf{PPKR}}$ stores at most one FILE record per cid (cf. CID3), **Uniqueness of** K is provided.

Retrieval Phase. The general structure of record keeping and interfaces of the retrieval phase are very similar to the initialization phase. First, the client has to start a retrieval session with the RETC interface, and the server has to agree in participating in the retrieval with the RETS interface, and there is again for each party an interfaces to let $\mathcal{F}_{\mathsf{PPKR}}$ output either success or failure to the parties. For the RETC interface, cid provides as input the password pw' it chose for this retrieval attempt. $\mathcal{F}_{\mathsf{PPKR}}$ records this in a RET record, again by either creating or updating the record (RC.1). It then outputs to $\mathcal{A}$ that cid started a retrieval session, where if the RETS interface was already called, the output also contains the bit $\mathsf{pw} \stackrel{?}{=} \mathsf{pw}'$, where pw is the password from the backup file that cid tries to retrieve (RC.2). We let $\mathcal{F}_{\mathsf{PPKR}}$ leak this information to the adversary because many protocols, including the WBP, leak via their communication pattern whether the client used the correct password or not. For example, a server might only send its last message to the client if it has previously learned that the client's password was correct.

For the RETS interface, the server can again supply the additional client identity cid', which acts exactly as in the initialization phase (RS.2) (**Client authentication** during retrieval). The server inputs further values pw^*, K^*, i, which become effective only if the server is fully corrupt. They are used to indicate which file or arbitrary values the server wants to use for the retrieval. This choice is determined as follows:

- If the server wants to run the retrieval with arbitrary values, it submits $\mathsf{pw}^* \neq \bot$ and $K^* \neq \bot$ (RS.3)
- If it wants to run the retrieval with a previously leaked file, it submits $\mathsf{pw}^* = \bot$. Using cid', it indicates the whose file it wants to use and the value i determines the exact LEAKED file.
- If it submits $\mathsf{pw}^* = \bot$ and $i = 0$ or if it is not fully corrupt, then $\mathcal{F}_{\mathsf{PPKR}}$ uses the currently stored file for cid (**key authenticity**)

After finding a file for a retrieval attempt, $\mathcal{F}_{\mathsf{PPKR}}$ checks if the retrieval attempt counter for cid has reached zero. In that case, it deletes the FILE record of cid to ensure that the key contained in that record cannot be recovered anymore (RS.4(2)) (**limited number of retrieval attempts with wrong passwords**). Otherwise, the retrieval attempt counter for cid' is decremented (RS.4(3)) and the password pw and key K obtained from the FILE record are written to the RET record (RS.5).

Finally, the functionality gives the output $(\text{RETS}, \mathsf{sid}, \mathsf{cid}', \mathsf{pw} \stackrel{?}{=} \mathsf{pw}')$ to the adversary $\mathcal{A}$ (RS.7), where the latter bit again refelcts that it may be leaked via the communication pattern whether the client used the correct password or not.

To complete the retrieval session, the adversary calls COMPLETERETC and

CompleteRetS, again with $\mathcal{F}_{\mathsf{PPKR}}$ enforcing no order in which the interfaces are called. Similarly to the the initialization phase, both interfaces verify that both parties gave their respective input (cf. CRC.1 and CRS.1) and both take an input b, which immediately let's the interface output Fail if $b = 0$ (cf. CRC.2 and CRS.2). Assuming $b = 1$, in CompleteRetC the functionality then compares the two password pw and pw' contained in the session state record and outputs the key K contained in the record to the client if they are the same (CRC.2 and CRC.3). The CompleteRetS interface works similarly, however it also outputs Succ to the server if the file was initialized by a malicious party, reflecting that they can reset their counter even when retrieving with wrong passwords (CRS.2 - CRC.4).

It can be seen from the outputs towards the server (CRS.4) and the adversary (RS.7) that only one bit of information about the password used by an honest client during retrieval (i.e., match or no match) is leaked, protecting the client from **online dictionary attacks**. Further, they contain no information about the key K, ensuring **secrecy of** K during retrieval.

Offline Attacks. The adversary has access to three interfaces MaliciousInit, MaliciousRet, and OfflineAttack to mount offline attacks. In the first two interfaces the adversary impersonates some client cid, however, as described in the discussion of the expected security properties, a client can only be impersonated if the server is corrupt. Therefore, both queries are ignored by $\mathcal{F}_{\mathsf{PPKR}}$ if S is honest (MR.1 and MI.1).

With the MaliciousInit interface, the adversary impersonates a client cid and executes a new initialization for cid. For this, $\mathcal{A}$ can choose a new password pw^* and a new key K^*, which are then stored in a new File record marked Stored that overwrites any existing File record for cid (MI.2). $\mathcal{F}_{\mathsf{PPKR}}$ resets the counter $\mathsf{ctrs}_{\mathsf{sid}}$ to 10 since with any new initialization the client gets 10 new retrieval attempts.

With the interface MaliciousRet the adversary impersonates a client and tries to recover a client's key from a guessed password. To this end, $\mathcal{A}$ inputs the client identity cid and a password guess pw^*. $\mathcal{F}_{\mathsf{PPKR}}$ retrieves the File record of cid marked Stored (MR.2) and checks if cid has any retrieval attempts left by checking if $\mathsf{ctrs}_{\mathsf{sid}}[\mathsf{cid}] = 0$. If cid has no retrieval attempts left, it deletes the File record of cid (**limited number of offline guesses**) and outputs $(\text{DelRec}, \mathsf{sid}, \mathsf{cid})$ to $\mathcal{A}$ to notify $\mathcal{A}$ that the record was deleted (MR.3). Otherwise, the functionality proceeds to check whether the guessed password pw^* matches the password pw from the File record. If the passwords match, the adversary gets access to the key stored in the File record, and otherwise $\mathcal{F}_{\mathsf{PPKR}}$ returns Fail to the adversary (MR.4).

The OfflineAttack interface can be used to guess the password of a leaked file offline. The adversary specifies the leaked file it wants to attack with the values cid and i, and obtains the key K from the file if its password guess was

correct (OA.1). Note that the adversary can start using the OFFLINEATTACK on records only *after* it compromised them, which ensures **pre-computation attack resistance**.

Differences between PPKR and 1-PPSS A password-protected secret sharing scheme [BJSL11] allows a user to retrieve a password-protected secret from a set of servers. The servers cannot derive or offline-attack the user's data unless a certain subset of them colludes. A PPKR scheme could be interpreted as a 1-PPSS scheme, i.e., where only one server is involved in storing and retrieving the password-protected secret. While both primitives are very similar considering only honest parties, it is actually the server corruption model that greatly differs for PPKR and 1-PPSS. Intuitively, the one server of a 1-PPSS scheme holds the only share of a secret (or key, in the terminology of PPKR), i.e., the full secret. If such a server is compromised, unlimited offline guesses on the shared user secret are unavoidable. PPKR is stronger: upon server compromise, only a *limited* number of password guesses are allowed on user secrets. Hence, PPKR never falls back to 1-PPSS, due to the stronger guarantees upon server compromise.

4.3.3 On Strengthening $\mathcal{F}_{\mathsf{PPKR}}$

We discuss two potential was of strengthening $\mathcal{F}_{\mathsf{PPKR}}$, both inspired by the discussions of 4.2.6. First, while $\mathcal{F}_{\mathsf{PPKR}}$ deletes keys of honest users whenever a user re-initializes (e.g., when refreshing the key, or when changing the password) as long as the server is honest, it does not guarantee uniqueness of clients' backup keys if the server is corrupted. Consequently, the **limited number of offline guesses** holds only *per initialized key* of a user, and not *per user*. The reason why we go with the weaker $\mathcal{F}_{\mathsf{PPKR}}$ is that the WBP cannot guarantee the stronger version, and thus we have to reflect this weakness for the security analysis of the actual protocol. However, as we demonstrate in Chapter 5, we can construct protocols that provide stronger security guarantees in this regard than WBP. For this reason, we state here the properties that we ideally demand from $\mathcal{F}_{\mathsf{PPKR}}$, and the corresponding functionality can be read from Figures 4.7 to 4.9 by dropping the code in dashed boxes. The security guarantees are then strengthened as follows.

- **Uniqueness of** K: ~~If the server is honest,~~ initialization of a key K by user cid deletes any key that cid previously initialized, *even if the server is malicious.*
- **Limited number of offline guesses**: The above guarantee extends to corrupt servers, i.e., after 10 wrong password guesses to recover ~~any honestly initialized~~ *the latest honestly initialized key* K *of any client*, K is deleted and cannot be retrieved by anyone anymore if no file containing K was leaked.
- **Client authentication**: As long as the server is ~~honest~~ *not fully corrupt*, only the client who initialized K can attempt to retrieve K. This must hold even if the password used during initialization becomes publicly known.

- **Key authenticity**: Let K be the key initialized by any client cid. Then, if cid retrieves with the correct password, it obtains K as long as the server is ~~honest~~ *not fully corrupt*.

Second, we can strengthen $\mathcal{F}_{\mathsf{PPKR}}$ by disallowing corrupt parties to reset the counter of files they initialized while using wrong passwords. For the same reasons as above, we have to define $\mathcal{F}_{\mathsf{PPKR}}$ including this weakness. The strengthened functionality can be read read from Figures 4.7 to 4.9 by dropping the code in solid boxes. The security guarantees are then strengthened as follows.

- **Limited number of retrieval attempts with wrong passwords**: Let K denote the key initialized by ~~an honest client cid~~ *any party* P, and assume that ~~cid~~ P later runs the retrieval phase 10 times consecutively with a wrong password. Then, unless the server is fully corrupt, it erases all K-dependent information, i.e., K cannot be retrieved anymore. This must hold even if the client becomes maliciously corrupted after initializing K.

4.4 Security Analysis

The Difficulty of a Security Analysis. The WBP relies on the strong asymmetric password-authenticated key exchange (saPAKE) protocol OPAQUE [JKX18], which comes with a security analysis in the UC framework. This immediately raises the question whether one could modularize the analysis and leverage the UC composition theorem to obtain our main result. The approach would be as follows:

1. Prove that OPAQUE UC-emulates functionality $\mathcal{F}_{\mathsf{saPAKE}}$ (proven already in [JKX18]).

2. Prove that the WBP using $\mathcal{F}_{\mathsf{saPAKE}}$ instead of OPAQUE UC-emulates $\mathcal{F}_{\mathsf{PPKR}}$ (presumably a simpler proof than proving security using OPAQUE).

3. Invoke the UC composition theorem: it yields that from 1. and 2. above it directly follows that WBP using OPAQUE UC-emulates $\mathcal{F}_{\mathsf{PPKR}}$.

However, this is not the road that we are able to take in this work. Instead, we have to prove the statement in item 3 above from scratch because of the following two main reasons. First, OPAQUE is a *key exchange* protocol that results in two parties sharing a key, while the goal of the WBP is to *hide the key from the server*. Second, the WBP deploys version 03 of the OPAQUE Internet Draft [KLW21] to which the security analysis of the OPAQUE framework [JKX18] does not apply, for a multitude of reasons that we describe in Appendix B.

The main challenge when performing a security analysis of the WBP is to tame its complexity. To this end, we modularize the security proofs of the underlying OPRF and the AKE schemes. Since previous security analyses did not apply to the protocol versions deployed by the WBP, we first show their security separately.

Then we use the resulting simulators as subcomponents of the WBP simulator. This proof technique, which was already used for AKE in [JKX18], avoids formulating the WBP in the AKE- and OPRF-hybrid model. The latter is not even possible, due to the non-black-box use that the WBP makes of these components.

To further tame the complexity, we only consider the Lev-1 security of the WBP and leave the analysis of Lev-2 and Lev-3 security as open problems. Nevertheless, even when proving only Lev-1 security, this provides confidence in the design choices that WhatsApp made for WBP, indeed resulting in a secure PPKR protocol. Moreover, having proven Lev-1 security, it seems likely that if an adversary wants to break the security of WBP in practice, this has to involve breaking the security of either the outsourced storage system or the HSM, leaving WhatsApp with only two potential points of failure that need to be protected.

4.4.1 Security of the 2HashDH OPRF

In this section we analyze the security of the 2HDH protocol that is used as a building block in the WBP. We first state a "multi-key" version of the OPRF functionality from [JKX18] that allows evaluation of PRFs with respect to many different keys, and where we drop the ability of the functionality to export a transcript prefix to the application. The reasons for these changes are as follows:

- **Multi-key setting:** WhatsApp's PPKR scheme in Figures 4.4 and 4.5 uses an OPRF called "2HashDH" [JKKX16] (see Figure 4.11 for the protocol description) where hash functions H_1 and KDF_2 do not have domains that are separated for different PRF keys. More concretely, if two clients initialize or retrieve using the same password pw, both compute the same value $H_1(\mathsf{pw})$ as a first step of the 2HashDH protocol. Hence, the security analysis of the OPRF part of WhatsApp's PPKR scheme cannot rely on any domain separation occurring.[16]

- **Dropping prefixes:** As we do not use $\mathcal{F}_{\mathsf{OPRF}}$ to formulate WhatsApp's PPKR scheme (i.e., we do not formulate it in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model) but only rely on the existence of a simulator for 2HashDH in the proof, we do not require the export of parts of the transcript in order to, e.g., sign or compare them. The OPRF functionality used in the analysis of OPAQUE [JKX18] had to introduce such exportation in order to be able to state the OPAQUE protocol in the OPRF-hybrid setting.

[16] We note that the security analysis of OPAQUE [JKX18] is carried out with respect to a single-key OPRF functionality and hence proves the security of OPAQUE only when hash domains of the two hash functions of 2HashDH are separated, e.g., by hashing unique session/key identifiers alongside the other inputs. However, OPAQUE in practice (e.g., [BKLW22]) does not deploy such domain separation, since the negotiation and memorization of session identifiers is expensive and partly contradicts with the purpose of the scheme being password-only. Hence, for a meaningful analysis of these deployed versions of OPAQUE, our multi-key version of $\mathcal{F}_{\mathsf{OPRF}}$ should be used.

We note that [JKX18] proves the security of 2HashDH without reliance on authenticated channels between the client and the server (previous works [JKKX16] still relied on such channels). This fits our setting, where the OPRF is run between the client and the HSM holding all PRF keys. Neither is the client authenticated to the HSM (it is only authenticated toward the server, but a malicious server can lie to the HSM about this authentication), nor can the client determine which key was used by the HSM (the malicious server can let the HSM use any of its PRF keys).

While WhatsApp's implementation of 2HashDH is in a strong setting where servers/PRF key holders are all played by the HSM and hence can be assumed incorruptible and uncompromisable (i.e., they will always follow the protocol, and they will never leak their PRF keys), we opt for a general treatment of OPRFs including server corruption and compromise in this section. While we do not require the analysis of these settings within this work, we believe that a general treatment and analysis of 2HashDH *without* domain separation has great relevance for other works [JKX18, BKLW22, DFHSW23].

Having summarized where we reuse results from, how we change them, and why, we now describe the technical contents of this section. In Figure 4.10 we state a multi-key OPRF functionality adopted from [JKX18]. In Figure 4.11 we give the multi-key version of the 2HashDH OPRF of [JKX18]. In Figure 4.12 we give the simulator that demonstrates that 2HashDH UC-realizes $\mathcal{F}_{\mathsf{OPRF}}$.

Multi-Key OPRF Model

Our functionality $\mathcal{F}_{\mathsf{OPRF}}$ closely follows the design from [JKX18], but we extend the functionality to be able to handle multiple servers and even multiple PRF keys per server. $\mathcal{F}_{\mathsf{OPRF}}$ is depicted in Figure 4.10 and we mark in gray the changes over [JKX18] that enable our $\mathcal{F}_{\mathsf{OPRF}}$ to handle multiple PRF keys and servers. We now explain the functionality's interfaces and parameters in detail.

The functionality $\mathcal{F}_{\mathsf{OPRF}}$ implements oblivious access to a truly random function family $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(\cdot) : \{0,1\}^* \to \{0,1\}^\ell$. The functions are parameterized by a global session identifier sid and parameters S, kid that can take arbitrary values and can be interpreted as taking the role of the PRF key. For each pair S, kid, a truly random function table is maintained.

The INIT interface. Any server S can call this interface to initialize a new PRF key with identifier kid. We let $\mathcal{F}_{\mathsf{OPRF}}$ ignore subsequent inputs of same key identifiers per server, which models that we expect key identifiers to be unique per server (e.g., they correspond to account names of clients where each client evaluates their "own" PRF, or they describe the purpose of the PRF that is evaluated with this key). Key identifiers are not kept secret (i.e., they are leaked to the adversary). For each newly initialized PRF key by server S with identifier kid, $\mathcal{F}_{\mathsf{OPRF}}$ stores a record $\langle \mathsf{S}, \mathsf{kid} \rangle$ and sets a counter $\mathsf{tx}[\mathsf{S}, \mathsf{kid}]$ to 0.

Functionality $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$

The functionality is parametrized by a PRF output-length ℓ. For every kid, x, value $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x)$ is initially undefined, and if an undefined value $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x)$ is referenced then $\mathcal{F}_{\mathsf{OPRF}}$ assigns $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x) \xleftarrow{\$} \{0,1\}^{\ell}$.

Initialization:
On (Init, sid, kid) from S, if this is the first Init message for kid, set $\mathsf{tx}[\mathsf{S}, \mathsf{kid}] = 0$, store $\langle \mathsf{S}, \mathsf{kid} \rangle$ and send (Init, sid, kid, S) to $\mathcal{A}$. Ignore all subsequent Init messages for kid from S. // Unique key identifiers per server.

Server Compromise:
On (Compromise, sid, kid, S) from $\mathcal{A}$, mark $\langle \mathsf{S}, \mathsf{kid} \rangle$ as Compromised. If S is corrupted, all key identifiers kid with records $\langle \mathsf{S}, \mathsf{kid} \rangle$ are marked as Compromised.
Note: Message (Compromise, sid, kid, S) *requires permission from the environment.* // Key-wise compromise is possible.

Offline Evaluation:
On (OfflineEval, sid, kid^*, S, x) from $\mathcal{A}$, send (OfflineEval, sid, kid^*, S, x, $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x)$) to $\mathcal{A}$ if any of the following hold: (i) $\langle \mathsf{S}, \mathsf{kid}^* \rangle$ is marked Compromised, (ii) $\mathsf{kid}^* = \mathsf{kid}$ for a kid previously received via the Init interface from S (iii) $\mathsf{kid}^* \neq \mathsf{kid}$ for all values kid previously received via the Init interface from S.

Evaluation:

- On (Eval, sid, kid, ssid, S, x) from $\mathsf{P} \in \{\mathsf{C}, \mathcal{A}\}$, record $\langle \mathsf{kid}, \mathsf{ssid}, \mathsf{P}, x \rangle$ and send (Eval, sid, kid, ssid, P, S) to $\mathcal{A}$.
- On (SndrComplete, sid, kid', ssid) from $\mathsf{P} \in \{\mathsf{S}', \mathcal{A}\}$:
  - Ignore the message if $\mathsf{P} = \mathsf{S}'$ is honest and there is no record $\langle \mathsf{S}', \mathsf{kid}' \rangle$. // Honest servers do not use unknown keys.
  - If $\mathsf{P} = \mathcal{A}$ then record $\langle \mathcal{A}, \mathsf{kid}' \rangle$ (if it does not exist already) // Adversary can play server with its own keys.
  - Increment $\mathsf{tx}[\mathsf{S}', \mathsf{kid}']$.
  - Send (SndrComplete, sid, kid', ssid, S′) to $\mathcal{A}$.
- On (RcvComplete, sid, kid^*, ssid, P, S*) from $\mathcal{A}$:
  - Ignore this message if there is no record $\langle *, \mathsf{ssid}, \mathsf{P}, x \rangle$ or if $\mathsf{tx}[\mathsf{S}^*, \mathsf{kid}^*] = 0$.
  - Decrement $\mathsf{tx}[\mathsf{S}^*, \mathsf{kid}*]$.
  - Send (EvalOut, sid, ssid, $F_{\mathsf{sid},\mathsf{S}^*,\mathsf{kid}^*}(x)$) to P.

Figure 4.10: A multi-key version of the ideal functionality $\mathcal{F}_{\mathsf{OPRF}}$ from [JKX18] (without prefixes). The ability to maintain multiple PRF keys is reflected in the addition of "key identifiers" kid, and we highlight the changes using gray boxes.

The Compromise **interface.** The effect of the compromise interface is that a record $\langle \mathsf{S}, \mathsf{kid} \rangle$ is marked Compromised. This corresponds to compromise of a PRF key, e.g., due to a breach at the server. We model key-wise compromise by letting the adversary specify which kid it wants to compromise at what server S, as it is possible that, e.g., a server only leaks keys that he recently has touched, while others remain securely stored. If a server gets corrupted (i.e., fully controlled by the adversary), all its keys are considered Compromised. Looking ahead, a key being marked compromised allows the adversary to evaluate the corresponding PRF an unbounded number of times (see the OfflineEval interface explanation below).

The OfflineEval **interface.** This interface can be used by the adversary to evaluate any of the random functions $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(\cdot)$ (identified by the "PRF key" S, kid) on any input x. However, $\mathcal{F}_{\mathsf{OPRF}}$ will only return the corresponding PRF value if the corresponding key is considered to be in the hands of the adversary. This is the case if S is corrupt or S, kid was already compromised, or if S, kid was never honestly initialized. If any of these checks pass, $\mathcal{F}_{\mathsf{OPRF}}$ returns $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x)$ to the adversary.

The Eval **interface.** This interface is called by any client C who wants to evaluate a specific PRF identified by S, kid on a secret input x. In order to allow for parallel evaluation sessions, the interface takes a subsession identifier ssid. $\mathcal{F}_{\mathsf{OPRF}}$ stores the request and informs the adversary, keeping input x private.

The SndrComplete **interface.** This interface allows a server to signal that it wants to assist in a specific evaluation identified by ssid, using the PRF key of that server identified by kid'. Note that $\mathcal{F}_{\mathsf{OPRF}}$ does not enforce the intended key identifier (specified in Eval input by the client) and the used key identifier (specified in SndrComplete input by the server) to be the same. This allows for the analysis of OPRF protocols that do not assume clients to be authenticated, and hence messages by clients can be modified. In particular the method of clients telling the server which key to use might not be tamper-proof. To proceed with the interface explanation, $\mathcal{F}_{\mathsf{OPRF}}$ allows server participation only if the corresponding key exists at a server (or the server is using a malicious key). $\mathcal{F}_{\mathsf{OPRF}}$ then increases the "evaluation ticket" counter $\mathsf{tx}[\mathsf{S}, \mathsf{kid}']$ by 1. Looking ahead, this counter will reflect the number of PRF evaluations that a server agreed to assist with, on a per-key basis.

The RcvComplete **interface.** Finally, the RcvComplete interface can be called by the adversary at any time to let a client C finalize an open evaluation session identified by ssid. $\mathcal{F}_{\mathsf{OPRF}}$ only continues the request if P is expecting to receive an output for ssid. The adversary has the freedom to specify with respect to which key $\mathsf{S}^*, \mathsf{kid}^*$ the client receives the evaluation for, with only one constraint:

there needs to be an evaluation ticket $\mathsf{tx}[\mathsf{S}^*, \mathsf{kid}^*]$. This ensures that evaluation of the PRF with respect to honest keys held by servers cannot happen more times than the corresponding server has agreed to assist in the evaluation. If an evaluation ticket for the specified key is found, it is taken away by decreasing the counter, and the PRF output is sent to P. We note that the flexibility of the adversary in $\mathcal{F}_{\mathsf{OPRF}}$ that lets it decide at the very latest point how to spend evaluation tickets is what has allowed to prove universal composability for efficient protocols such as 2HashDH [JKKX16] in the past, as it allows for "late extraction" of adversarial PRF keys.

Security of Multi-Key 2HashDH

The multi-key version of 2HDH can be found in Figure 4.11. The changes over the single-key version are quite minimal: essentially all interfaces receive a key identifier kid as additional input. The client who wants to evaluate a PRF with key identifier kid informs the server about it by sending kid alongside the first message. The server who receives the first message takes the kid and looks up the PRF key according to this identifier. There is no authenticity of key identifiers if channels are not client-authenticated, as in the setting of the WhatsApp backup protocol.

We stress that the addition of key identifiers results in a subtle but crucial difference: in multi-key 2HDH (Figure 4.11), hash function domains are not separated for each key identifier kid. That is, Alice's computation to evaluate her PRF identified by $\mathsf{kid}_{\mathsf{Alice}}$ at input x involves computation of $H_1(x)$, which is the same value that Bob computes if he wants to evaluate his PRF identified by $\mathsf{kid}_{\mathsf{Bob}}$ at the same input x. Without client-authenticated channels, the adversary can hence maliciously "reroute" PRF evaluation to different keys. More specifically, Alice computes $H_1(x)^r$ and sends $(\mathsf{kid}_{\mathsf{Alice}}, H_1(x)^r)$ to her server. The adversary rewrites this message to $(\mathsf{kid}_{\mathsf{Bob}}, H_1(x)^r)$ such that the server applies Bob's key and sends back to Alice the value $(H_1(x)^r)^{k_{\mathsf{Bob}}}$. Alice removes the blinding factor r and outputs $H_2(x, H_1(x)^{k_{\mathsf{Bob}}})$, which is an evaluation of x of Bob's PRF. Note that Alice cannot notice that she computed somebody else's PRF.[17]

Theorem 3. *Let $H_1 : \{0,1\}^* \to \mathbb{G}$ be a hash function into a group of order $q \in \mathbb{N}$, $H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\ell$ with $\ell \in \mathbb{N}$ be another hash function, and let $k \xleftarrow{\$} \mathbb{Z}_q$. Suppose the (N, Q) one-more DH assumption holds for $\mathbb{G}$, where $Q := q_E$ is the maximum number of $(\textsc{Eval}, *, \mathsf{kid}, \mathsf{S}, *)$ queries over all tuples $(\mathsf{kid}, \mathsf{S})$ made by the environment $\mathcal{Z}$, $N := q_E + q_H$, and q_H is the total number of H_1 queries made by $\mathcal{Z}$. Then the "multi-key" protocol 2HDH of Figure 4.11 UC-realizes the "multi-key" functionality $\mathcal{F}_{\mathsf{OPRF}}$ of Figure 4.10, with hash functions H_1, H_2 modeled as random oracles.*

[17] We note that the analysis of OPAQUE [JKX18], which is carried out using multiple instances of single-key 2HashDH, does not examine the effect of this attack since the parallel execution of many single-key 2HashDH instances introduces domain separation into all random oracles.

2HashDH(H_1, H_2)

Components:
Hash functions $H_1 : \{0,1\}^* \to \mathbb{G}, H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\ell$ with $\ell \in \mathbb{N}$ and $\mathbb{G} = \langle g \rangle$ a group of order q. The two hash functions are specified for one particular sid, which has to be folded into the input.

Initialization:
On $(\textsc{Init}, \mathsf{sid}, \boxed{\mathsf{kid}})$, the server S picks $k \xleftarrow{\$} \mathbb{Z}_q$ and stores $\langle \mathsf{sid}, \boxed{\mathsf{kid}}, k \rangle$.

Server Compromise:
On $(\textsc{Compromise}, \mathsf{sid}, \boxed{\mathsf{kid}})$, if there is a record $\langle \mathsf{sid}, \boxed{\mathsf{kid}}, k \rangle$, the server S reveals k to the adversary.

Offline Evaluation:
On $(\textsc{OfflineEval}, \mathsf{sid}, \boxed{\mathsf{kid}}, x)$, the server retrieves record $\langle \mathsf{sid}, \boxed{\mathsf{kid}}, k \rangle$ and outputs $(\textsc{OfflineEval}, \mathsf{sid}, \boxed{\mathsf{kid}}, H_2(x, H_1(x)^k))$.

Online Evaluation:

- On $(\textsc{Eval}, \mathsf{sid}, \boxed{\mathsf{kid}}, \mathsf{ssid}, \mathsf{S}', x)$ the client C picks $r \xleftarrow{\$} \mathbb{Z}_q$, records $\langle \mathsf{sid}, \mathsf{ssid}, r \rangle$ sends $(\mathsf{sid}, \boxed{\mathsf{kid}}, \mathsf{ssid}, H_1(x)^r)$ to the server S'.
- On $(\textsc{SndrComplete}, \mathsf{sid}, \boxed{\mathsf{kid}}, \mathsf{ssid})$ and message $(\mathsf{sid}, \boxed{\mathsf{kid}}, \mathsf{ssid}, a)$ from C, s.t. $a \in \mathbb{G}$, the server S retrieves record $\langle \mathsf{sid}, \boxed{\mathsf{kid}}, k \rangle$ and sends $(\mathsf{sid}, \mathsf{ssid}, a^k)$ to C.
- On $(\mathsf{sid}, \mathsf{ssid}, b)$ s.t. $b \in \mathbb{G}$, the client C retrieves record $\langle \mathsf{sid}, \mathsf{ssid}, r \rangle$, aborts if the tuple is not found and else outputs $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, H_2(x, b^{1/r}))$.

Figure 4.11: The multi-key version of protocol 2HashDH that realizes $\mathcal{F}_{\mathsf{OPRF}}$. The changes introduced over [JKX18] due to the handling of multiple PRF keys are marked with gray boxes, and the exportation of prefixes is dropped.

More precisely, for any adversary against 2HashDH, there is a simulator $\text{SIM}_{\mathsf{2HDH}}$ *that interacts with* $\mathcal{F}_{\mathsf{OPRF}}$ *and produces a view that no environment* $\mathcal{Z}$ *can distinguish with advantage better than*

$$\mathbf{Adv}^{\mathcal{F}_{\mathsf{OPRF}}}_{\mathsf{2HDH},\text{SIM}_{\mathsf{2HDH}},\mathcal{Z}}(\lambda) \leq q_I \mathbf{Adv}^{\mathsf{OMDH}}_{\mathbb{G},q_E+q_H,\mathcal{A}}(\lambda)(\lambda) + \frac{(q_E+q_H)^2}{q}$$

where q_I *is the number of honestly initialized keys in the system.*

Proof intuition. There is already a proof of universal composability of single-key 2HashDH in [JKX18], by giving an algorithm that simulates the protocol run if there is only one honest PRF key. If there are several honest PRF keys, the question to resolve in our proof is whether the individual simulators from [JKX18] can be orchestrated to run in parallel, without any clashes in programming the random oracles. The idea to avoid clashes is the following: first, due to the high entropy in PRF keys, a clash in H_2 programming is unlikely to occur, since k is part of the H_2 inputs. Inputs to H_1 are the values at which a PRF should be evaluated, and hence they can coincide (e.g., different users have the same passwords that they need to feed into their PRF). However, the single-key simulator of [JKX18] does not rely on programming H_1 outputs to values specific to a certain PRF key, but rather relies on *knowledge of a trapdoor* of the hash output. Our multi-key simulator can thus apply the following strategy: it first chooses trapdoors itself and plants them into H_1 outputs, and then it runs the individual simulators on these joint trapdoors. The precise code of our simulator is given in Figure 4.12 and the detailed proof follows below.

Proof. We argue that $\text{SIM}_{\mathsf{2HDH}}$ of Figure 4.12 generates a view to an arbitrary environment $\mathcal{Z}$ that is indistinguishable from $\mathcal{Z}$'s interaction with the real world where parties run protocol 2HashDH of Figure 4.11. Without loss of generality, suppose $\mathcal{A}$ is the dummy adversary [Can01] who merely passes through all its messages to and from $\mathcal{Z}$. The interfaces and view of $\mathcal{Z}$ are as follows:

- Client: $\mathcal{Z}$ sends $(\text{EVAL}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{S}, x)$ to a client and eventually receives back a PRF value $(\text{EVALOUT}, \mathsf{sid}, \mathsf{ssid}, y)$ under the key identified by kid, held by S.

- Server: $\mathcal{Z}$ sends $(\text{SNDRCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ to a server in order to let that server finish session ssid using key identifier kid. $\mathcal{Z}$ does not expect to see any output from the server upon sending this input.

- Adversary: $\mathcal{Z}$ expects to receive both protocol messages from $\mathcal{A}$ in case client and server are honest. $\mathcal{Z}$ sends $(\mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, a)$ to $\mathcal{A}$ as any client's message, and $(\mathsf{sid}, \mathsf{ssid}, b)$ as any server's message. Observe that, since 2HashDH is run over unauthenticated channels, such messages can be introduced by $\mathcal{Z}$ without corrupting anybody.

- Random oracles: $\mathcal{Z}$ can query both $H_1(x)$ and $H_2(y, z)$ to $\mathcal{A}$ for any values x, y, z.

We now describe the above view of $\mathcal{Z}$ in more detail and for both worlds. For any chosen kid, x, $\mathcal{Z}$ receives transcript values a, b from $\mathcal{A}$ for corresponding honest clients and servers, and a PRF value $(\textsc{EvalOut}, \mathsf{sid}, \mathsf{ssid}, y)$ in case the client is honest. In the real execution, we have $a = H(x)^r$ for a randomly chosen $r \in \mathbb{Z}_q$, $b = a^k$ for a randomly chosen $k \in \mathbb{Z}_q$, $y = H_2(x, H_1(x)^k)$, and queries to H_1 and H_2 answered consistently and uniformly at random. In the ideal execution, we have $a = g_J$ a randomly chosen group element, $b = g_J^{k'}$ for a randomly chosen $k' \xleftarrow{\$} \mathbb{Z}_q$, $y \xleftarrow{\$} \{0,1\}^\ell$ as chosen by $\mathcal{F}_{\mathsf{OPRF}}$, and H_1 and H_2 values answered with uniform values from the appropriate ranges. We now argue indistinguishability of both worlds in detail.

- Message a ($H(x)^r$ vs. g_J): since the uniformly chosen $r \in \mathbb{Z}_q$ is never revealed by an honest client, $H(x)^r$ is uniformly random to $\mathcal{Z}$ and hence indistinguishable from g_J.
- Message b (a^k vs. $a^{k'}$): k is chosen at random by the honest server and not revealed to $\mathcal{Z}$, while k' is chosen at random by $\textsc{Sim}_{\mathsf{2HDH}}$, and not revealed to $\mathcal{Z}$. Hence both a^k and $a^{k'}$ are indistinguishable for $\mathcal{Z}$.
- Output $(\textsc{EvalOut}, \mathsf{sid}, \mathsf{ssid}, y)$ ($H_2(x, H_1(x)^k)$ vs. $F_{\mathsf{sid},i}(x)$): the only way to distinguish the real world y from the ideal world is to query $(x, H_1(x)^k)$ to the H_2 oracle. However, $\textsc{Sim}_{\mathsf{2HDH}}$ is able to detect this: if k is the key used by an honest server, then there is a record $\langle F, *, \mathsf{kid}, k, u^{1/r}\rangle$ for the corresponding r from the H_1 record of x (cf. first bullet of step 8 in Figure 4.12). Hence in this case $\textsc{Sim}_{\mathsf{2HDH}}$ learns the key identifier kid which this H_2 query of $\mathcal{Z}$ is consistent with. If on the other hand k is a key already used by a corrupt server/the network adversary, then $\textsc{Sim}_{\mathsf{2HDH}}$ has a record $\langle M, \mathcal{A}, i, \bot, u^{1/r}\rangle$ for the corresponding r from the H_1 record of x (cf. second bullet of step 8 in Figure 4.12). Hence also in this case $\textsc{Sim}_{\mathsf{2HDH}}$ learns the key identifier i. If any key identifier is found, $\textsc{Sim}_{\mathsf{2HDH}}$ obtains the correct PRF value $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x)$ (or $F_{\mathsf{sid},\mathcal{A},i}(x)$) from $\mathcal{F}_{\mathsf{OPRF}}$ via the $\textsc{Eval}$ (or $\textsc{OfflineEval}$) interface (depending on whether the server holding the key identifier is compromised/corrupt or honest), and sets it to be equal to $H_2(x, u)$. Hence, if $\mathcal{F}_{\mathsf{OPRF}}$ replies with a value, the outputs are equal. If not, $\textsc{Sim}_{\mathsf{2HDH}}$ aborts and we analyze the probability for that happening below. Note that this also ensures that $H_2(x, u)$ equals an $\textsc{OfflineEval}$ query of an honest server for one of its own kid.
- Random oracle H_1: since g_J in step 4 of $\textsc{Sim}_{\mathsf{2HDH}}$ is chosen at random, the simulated responses are indistinguishable from the ones chosen by the random oracle in the real protocol.
- Random oracle H_2: $\textsc{Sim}_{\mathsf{2HDH}}$ programs H_2 to either a uniform value (cf. third bullet of step 8 in Figure 4.12) or to an output of $\mathcal{F}_{\mathsf{OPRF}}$, which is itself chosen by $\mathcal{F}_{\mathsf{OPRF}}$ uniformly at random. Hence, the H_2 outputs of $\textsc{Sim}_{\mathsf{2HDH}}$ are equally distributed to the outputs of the random oracle H_2 in the real

world.

It is left to analyze the probability that $\text{SIM}_{\mathsf{2HDH}}$ queries $\mathcal{F}_{\mathsf{OPRF}}$ with either $(\text{EVAL}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{S}, x)$ and $(\text{RCVCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathcal{A}, \mathsf{S})$, or with $(\text{OFFLINEEVAL}, \mathsf{sid}, i, \mathsf{S}', x)$ and does not receive a PRF value as reply. This can happen for EVAL and RCVCOMPLETE queries in case their inputs do not correspond to each other, or if no tickets are left (i.e., $\mathsf{tx}[\mathsf{S}, \mathsf{kid}] = 0$). For OFFLINEEVAL, $\text{SIM}_{\mathsf{2HDH}}$ only does not receive a reply if S' is honest and has previously initialized key identifier i.

For OFFLINEEVAL, $\text{SIM}_{\mathsf{2HDH}}$ calls this interface with inputs S, i in three places in step 8, where in the first occurrence S is COMPROMISED, and in the second and third occurrence $\mathsf{S} = \mathcal{A}$. Thus, OFFLINEEVAL always outputs a PRF value y to $\text{SIM}_{\mathsf{2HDH}}$.

For EVAL and RCVCOMPLETE, SIM calls these interfaces in step 8, first bullet, second dash. Since $\text{SIM}_{\mathsf{2HDH}}$ uses corresponding inputs, $\mathcal{F}_{\mathsf{OPRF}}$ not replying is not due to mismatching inputs but due to $\mathsf{tx}[\mathsf{S}, \mathsf{kid}] = 0$ as checked by $\mathcal{F}_{\mathsf{OPRF}}$ in RCVCOMPLETE. Let $\text{FAIL}(\mathsf{S}, \mathsf{kid})$ denote the event that a $(\text{RCVCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathcal{A}, \mathsf{S})$ message is ignored. We have $\Pr[\text{SIM}_{\mathsf{2HDH}} \text{ aborts}] \leq \sum_{\mathsf{S},\mathsf{kid}} \Pr[\text{FAIL}(\mathsf{S}, \mathsf{kid})]$.

We now upper bound $\Pr[\text{FAIL}(\mathsf{S}, \mathsf{kid})]$ by reducing to the one-more DH problem using the reduction in Figure 4.13. The overall strategy of the reduction is the following: the challenge key $\bar{k}$ is only implicitly known as $g^{\bar{k}}$ and the reduction records the tuple $(g, g^{\bar{k}})$ as key of $\tilde{\mathsf{S}}, \tilde{\mathsf{kid}}$. The reduction puts the challenge generators $g_1, \ldots, g_N$ as H_1 replies and first messages of EVAL queries for $\tilde{\mathsf{S}}, \tilde{\mathsf{kid}}$. It is now left to run the rest of the execution without knowledge of $\bar{k}$ and the exponents (trapdoors) of the generators. The strategy is as follows:

- The reduction uses its $(\cdot)^{\bar{k}}$ exponentiation oracle to produce messages on behalf of $\tilde{\mathsf{S}}$ for key $\tilde{\mathsf{kid}}$.
- The reduction uses its DDH oracle $\mathsf{DDH}(g, g^{\bar{k}}, X, Y)$ to recognize an adversarially-given tuple (X, Y) that lets it win the one-more DH game.
- The reduction uses its DDH oracle $\mathsf{DDH}(g_j, Y, g_{j'}, B)$ to recognize re-usage of adversarial keys k in two evaluation transcripts (g_j, Y), $(g_{j'}, B)$.

From the reduction code in Figure 4.13 we can see the following.

- Every time the exponentiation oracle is used (step 6), a $(\text{SNDRCOMPLETE}, \mathsf{sid}, \mathsf{kid}, *)$ query was issued by S and hence $\mathsf{tx}[\mathsf{S}, \mathsf{kid}]$ was increased by 1.
- The counter $\mathsf{tx}[\mathsf{S}, \mathsf{kid}]$ is decreased whenever $(\text{RCVCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, *, \mathsf{S})$ is sent to $\mathcal{F}_{\mathsf{OPRF}}$, which happens in the first bullets of steps 7 and 8. It can be seen from the DDH oracle inputs that in both cases the adversary gave a tuple $g_j, g_j^{\bar{k}}$ (with $\bar{k}$ being the challenge key) and g_j is from $g_1, \ldots, g_N$.

Thus, if $\text{FAIL}(\mathsf{S}, \mathsf{kid})$ occurs and the reduction has guessed the correct initialization query, the number of such tuples is one more than the number of oracle queries

made by the reduction (assuming there is no collision in $g_1, \ldots, g_N$). That is,

$$\Pr[\textsc{Fail}(\mathsf{S}, \mathsf{kid}) \mid \text{no collision in } g_1, \ldots, g_N] \leq q_I \mathbf{Adv}^{\mathsf{OMDH}}_{\mathbb{G},n,\mathcal{A}}(\lambda)$$

with $n := q_E + q_H$, where q_I is the number of INIT queries (i.e., honestly initialized keys in the system), q_E is the maximum number of EVAL queries over all PRF keys, and q_H is the number of H_1 queries made by $\mathcal{Z}$.

On the other hand, the probability that there is a collision in $g_1, \ldots, g_N$ is upper bounded by N^2/q. Thus we have

$$\Pr[\textsc{Fail}] \leq q_I \mathbf{Adv}^{\mathsf{OMDH}}_{\mathbb{G},q_E+q_H,\mathcal{A}}(\lambda) + (q_E + q_H)^2/q.$$

□

4.4.2 Security of the 3DH AKE

In this section we analyze the security of the 3DH protocol as used in the WBP and the OPAQUE draft [KLW21]. In previous work [GJK21] it was shown that a slightly different version of 3DH UC-realizes key-hiding AKE (KH-AKE). Even though the two versions differ only in small details, the variant of 3DH used in WBP is unfortunately not covered by the analysis of [GJK21]. To close this gap, we reenact the previous analysis of [GJK21] for the 3DH variant as used in WBP and show that it UC-realizes AKE with security against key compromise impersonation (KCI), which is a security notion related to KH-AKE but which is not known to be implied by KH-AKE. AKE-KCI is used in [JKX18] to prove security of OPAQUE and we use it in Section 4.4.3 to prove that the WBP UC-realizes PPKR. The result shown in this section further justifies the decision to use 3DH as the AKE in the OPAQUE draft [KLW21] instead of HMQV, as suggested by [JKX18].

We consider the 3DH variant described in Figure 4.14, as matches the use in WBP. The difference from the variant considered in [GJK21] lies in the computation of the key k. While in [GJK21] it is computed as $H(\mathsf{sid}, \mathsf{P}, \mathsf{P}', X, Y, Z)$, in Figure 4.14 it is computed as $H(\mathsf{aux}, X, Y, Z)$, where aux is an arbitrary auxiliary input string. This reflects real-world scenarios, where often the session key is computed dependent on additional context information. This is also the case in WBP, where shk depends on the transcript pre. Note that we set aux to $\perp$ when creating a new session, since in many applications the full context information may not be available yet. This is also the case in WBP, where pre contains many values that are computed by the server and therefore unknown to the client at the start of the retrieval phase.

We introduce two small changes to the functionality $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ presented in [JKX18] and display the modified version of $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ in Figure 4.15. First, we add an auxiliary input aux from the adversary to the NEWKEY interface and ensure that NEWKEY only outputs the same session key for two sessions if they are

Simulator $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, N)$

The simulator obtains as input a session identifier sid indicating which (multi-key) $\mathcal{F}_{\mathsf{OPRF}}$ instance it communicates with, the description of two hash functions $H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^l, H_1 : \{0,1\}^* \to \mathbb{G}$ with $l \in \mathbb{N}$ and $\mathbb{G} = \langle g \rangle$ a group of order q, and a number $N \in \mathbb{N}$.

1. Pick and record N random numbers $r_1, \ldots, r_N \in \mathbb{Z}_q$ and set $g_1 \coloneqq g^{r_1}, \ldots, g_N \coloneqq g^{r_N}$. Set counter $J \coloneqq 1$ and $I \coloneqq 1$.
2. On $(\textsc{Init}, \mathsf{sid}, \mathsf{kid}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{OPRF}}$, record $\langle \mathsf{S}, \mathsf{kid} \rangle$ and $\langle F, \mathsf{S}, \mathsf{kid}, k, z = g^k \rangle$ for $k \xleftarrow{\$} \mathbb{Z}_q$.
3. On $(\textsc{Compromise}, \mathsf{sid}, \mathsf{kid}, \mathsf{S})$ from $\mathcal{A}$, mark $\langle \mathsf{S}, \mathsf{kid} \rangle$ as $\textsc{Compromised}$. Retrieve $\langle F, \mathsf{S}, *, \mathsf{kid}, k, * \rangle$ and send $(\textsc{Compromise}, \mathsf{sid}, \mathsf{kid})$ to $\mathcal{F}_{\mathsf{OPRF}}$ and $(\mathsf{sid}, \mathsf{kid}, k)$ to $\mathcal{A}$.
4. For each fresh query x to $H_1(\cdot)$, answer it with g_J and record $\langle H_1, x, r_J \rangle$. Set $J \coloneqq J + 1$.
5. Upon receiving $(\textsc{Eval}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{OPRF}}$, send $(\mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, g_J)$ to $\mathcal{A}$ as C's message to S and record $\langle \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, r_J \rangle$. Set $J \coloneqq J + 1$.
6. Upon receiving $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ from $\mathcal{F}_{\mathsf{OPRF}}$ and $(\mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, a)$ from $\mathcal{A}$ as some client's C message to some honest server S: if there is a record $\langle F, \mathsf{S}, \mathsf{kid}, k, * \rangle$, then send $(\mathsf{sid}, \mathsf{ssid}, a^k)$ as the response of S for client C to $\mathcal{A}$.
7. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, b)$ with $b \in \mathbb{G}$ from $\mathcal{A}$ as some server's S' message to a client C, retrieve record $\langle *, \mathsf{ssid}, \mathsf{C}, r \rangle$ and let g_j denote the message sent in step 5 for $\mathsf{ssid}, \mathsf{C}$.
   - [$\mathcal{A}$ delivers honestly.] If there is a record $\langle F, \mathsf{S}, \mathsf{kid}, k, * \rangle$ with $b = g_j^k$ and $\langle \mathsf{S}, \mathsf{kid} \rangle$ is not marked $\textsc{Compromised}$, send $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ plays server using non-fresh adversarial key.] If a record $\langle M, \mathcal{A}, i, \bot, b^{1/r} \rangle$ exists, send $(\textsc{SndrComplete}, \mathsf{sid}, i, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, i, \mathsf{ssid}, \mathsf{C}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ plays server with compromised key.] If there is a record $\langle F, \mathsf{S}, \mathsf{kid}, *, b^{1/r} \rangle$ and record $\langle \mathsf{S}, \mathsf{kid} \rangle$ is marked $\textsc{Compromised}$, send $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ uses fresh key.] If there is no such record $\langle T, *, *, *, b^{1/r} \rangle$, set $i \coloneqq I$, record $\langle M, \mathcal{A}, i, \bot, b^{1/r} \rangle$, and set $I \coloneqq I + 1$. Send $(\textsc{SndrComplete}, \mathsf{sid}, i, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, i, \mathsf{ssid}, \mathsf{C}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
8. For each fresh query (x, u) to $H_2(\cdot, \cdot)$, retrieve record $\langle H_1, x, r \rangle$. If there is no such record, then pick $H_2(x, u) \xleftarrow{\$} \{0,1\}^l$. Otherwise, do the following:
   - [$u = H(x)^k$ for a server's key.] If a record $\langle F, \mathsf{S}, \mathsf{kid}, k, z \rangle$ satisfies $z = u^{1/r}$, do:
     - [Compute PRF value for k, x offline.] If $\langle \mathsf{S}, \mathsf{kid} \rangle$ is $\textsc{Compromised}$ or S is corrupt, send $(\textsc{OfflineEval}, \mathsf{sid}, \mathsf{kid}, \mathsf{S}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, and on response $(\textsc{OfflineEval}, \mathsf{sid}, \mathsf{kid}, \mathsf{S}, x, y)$, set $H_2(x, u) \coloneqq y$.
     - [Compute PRF value for k, x online, relying on a ticket $\mathsf{tx}[\mathsf{S}, \mathsf{kid}]$.] If S is not $\textsc{Compromised}$, pick a fresh ssid^* and send $(\textsc{Eval}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}^*, \mathsf{S}, x)$, $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$, and $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}^*, \mathcal{A}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{OPRF}}$. If $\mathcal{F}_{\mathsf{OPRF}}$ ignores the last message, abort. Else, on $\mathcal{F}_{\mathsf{OPRF}}$'s response $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}^*, y)$, set $H_2(x, u) \coloneqq y$.
   - [$u = H(x)^k$ for an adversarial k.] Else, if there is a tuple $\langle M, \mathcal{A}, i, \bot, u^{1/r} \rangle$, send $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, and on response $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x, y)$ set $H_2(x, u) \coloneqq y$
   - [Fresh adversarial key.] Else, record $\langle M, \mathcal{A}, i, \bot, u^{1/r} \rangle$ for $i = I$, send send $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, and on response $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x, y)$ set $H_2(x, u) \coloneqq y$ and $I \coloneqq I + 1$.

Figure 4.12: The simulator that demonstrates that "multi-key" 2HashDH UC-realizes our "multi-key" $\mathcal{F}_{\mathsf{OPRF}}$. The simulator is adopted from [JKX18], Figure 14. We add key identifiers and run one instance of their simulator (without prefix simulation) per kid.

Reduction $\mathcal{R}(\tilde{\mathsf{S}}, \tilde{\mathsf{kid}}, g, y = g^{\bar{k}}, g_1, \ldots, g_N)$

The reduction runs simulator $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, \bot, N)$ with the following modifications. If a step is not listed, then there are no changes compared to $\textsc{Sim}_{\mathsf{2HDH}}$

2. [Place the challenge key:] On $(\textsc{Init}, \tilde{\mathsf{S}}, \tilde{\mathsf{kid}}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{OPRF}}$, record $\langle F, \tilde{\mathsf{S}}, \tilde{\mathsf{kid}}, \bot, y\rangle$ and $\langle \mathsf{S}, \tilde{\mathsf{kid}}\rangle$. From now on, use $\tilde{\mathsf{kid}}$ to denote the guessed key identifier, and $\tilde{\mathsf{S}}$ to denote the server who holds it. For all other Init queries, change the recorded tuple to format $\langle F, \mathsf{S}, \mathsf{kid}, k, (g, g^k)\rangle$.
4. [Put challenges in H_1:] As $\textsc{Sim}_{\mathsf{2HDH}}$, except that $\mathcal{R}$ records $\langle H_1, x, g_J\rangle$ instead of $\langle H_1, x, g_J\rangle$.
5. [Put challenges in Eval queries:] Upon $(\textsc{Eval}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \tilde{\mathsf{S}})$ with $\mathsf{kid} = \tilde{\mathsf{kid}}$ from $\mathcal{F}_{\mathsf{OPRF}}$, send $(\mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, g_J)$ to $\mathcal{A}$ as C's message to $\tilde{\mathsf{S}}$ and record $\langle \tilde{\mathsf{kid}}, \mathsf{ssid}, \mathsf{C}, \bot\rangle$. Set $J \coloneqq J + 1$. For $\mathsf{kid} \neq \tilde{\mathsf{kid}}$, there is no change in code here.
6. Upon receiving $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{OPRF}}$ and $(\mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, a)$ from $\mathcal{A}$ as some client's C message to some honest server S:
   - If $(\mathsf{S}, \mathsf{kid}) \neq (\tilde{\mathsf{S}}, \tilde{\mathsf{kid}})$, $\mathcal{R}$ acts like $\textsc{Sim}_{\mathsf{2HDH}}$.
   - [Use exponentiation oracle to compute b for key $\tilde{\mathsf{kid}}$:] If $(\mathsf{S}, \mathsf{kid}) = (\tilde{\mathsf{S}}, \tilde{\mathsf{kid}})$ then send b to the exponentiation oracle to receive back $b^{\bar{k}}$, and send $(\mathsf{sid}, \mathsf{ssid}, b^{\bar{k}})$ as the response of S for client C to $\mathcal{A}$. Record $(\mathsf{reftuple}, a, b^{\bar{k}})$ if this was the first usage of the oracle.
7. [Use DDH oracle to compensate for not knowing H_1 exponent r:] Upon receiving $(\mathsf{sid}, \mathsf{ssid}, b)$ with $b \in \mathbb{G}$ from $\mathcal{A}$ as some server's S' message to a client C, retrieve records $\langle *, \mathsf{ssid}, \mathsf{C}, r\rangle$ and $(\mathsf{reftuple}, A, B)$, let g_j denote the message sent in step 5 for $\mathsf{ssid}, \mathsf{C}$, and do:
   - [$\mathcal{A}$ delivers b for challenge $\tilde{\mathsf{kid}}$ honestly.] If $\mathsf{DDH}(g_j, b, A, B) = 1$, send $(\textsc{RcvComplete}, \mathsf{sid}, \tilde{\mathsf{kid}}, \mathsf{ssid}, \mathsf{C}, \tilde{\mathsf{S}})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ delivers b for non-challenge kid honestly.] If there is a record $\langle F, \mathsf{S}, \mathsf{kid}, k, *\rangle$ with $b = g_j^k$ and record $\langle \mathsf{S}, \mathsf{kid}\rangle$ is not marked Compromised, send $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ plays server using non-fresh adversarial key.] If there is a record $\langle M, \mathcal{A}, i, \bot, (G, H)\rangle$ with $\mathsf{DDH}(G, H, g_j, b) = 1$, send $(\textsc{SndrComplete}, \mathsf{sid}, i, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, i, \mathsf{ssid}, \mathsf{C}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ plays server with compromised key.] If there is a record $\langle F, \mathsf{S}, \mathsf{kid}, k, (g, g^k)\rangle$ with $b = g_j^k$ and record $\langle \mathsf{S}, \mathsf{kid}\rangle$ is marked Compromised, send $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   - [$\mathcal{A}$ uses fresh key.] In any other case, set $i \coloneqq I, I \coloneqq I + 1$, record $\langle M, \mathcal{A}, i, \bot, (g_j, b)\rangle$, send $(\textsc{SndrComplete}, \mathsf{sid}, i, \mathsf{ssid})$ and $(\textsc{RcvComplete}, \mathsf{sid}, i, \mathsf{ssid}, \mathsf{C}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
8. For each fresh query (x, u) to $H_2(\cdot, \cdot)$, retrieve record $\langle H_1, x, g_j\rangle$. If there is no such record, then pick $H_2(x, u) \xleftarrow{\$} \{0,1\}^l$. Otherwise, retrieve $\langle \mathsf{reftuple}, A, B\rangle$ and do the following:
   - [$u = H(x)^k$ for the challenge key.] If $\mathsf{DDH}(g_j, u, A, B)$ then pick a fresh identifier ssid^* and send $(\textsc{Eval}, \mathsf{sid}, \tilde{\mathsf{kid}}, \mathsf{ssid}^*, \bot, x)$ and $(\textsc{RcvComplete}, \mathsf{sid}, \tilde{\mathsf{kid}}, \mathsf{ssid}^*, \mathcal{A}, \tilde{\mathsf{S}})$ to $\mathcal{F}_{\mathsf{OPRF}}$. If $\mathcal{F}_{\mathsf{OPRF}}$ ignores the last message then abort. Else, on $\mathcal{F}_{\mathsf{OPRF}}$'s response $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}^*, y)$, set $H_2(x, u) \coloneqq y$.
   - [$u = H(x)^k$ for any other server's key.] If some $\langle F, \mathsf{S}, \mathsf{kid}, k, (g, g^k)\rangle$ satisfies $u = g_j^k$, then proceed as $\textsc{Sim}_{\mathsf{2HDH}}$ in the case that some $\langle F, \mathsf{S}, \mathsf{kid}, k, z\rangle$ satifies $z = u^{1/r}$.
   - [$u = H(x)^k$ for an adversarial k.] Else, if there is a tuple $\langle M, \mathcal{A}, i, \bot, (G, H)\rangle$ with $\mathsf{DDH}(G, H, g_j, u) = 1$ then send $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, and on response $(\textsc{OfflineEval}, \mathsf{sid}, i, \mathcal{A}, x, y)$ set $H_2(x, u) \coloneqq y$
   - [Fresh adversarial key.] Else, send $(\textsc{OfflineEval}, \mathsf{sid}, I, \mathcal{A}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, on response $(\textsc{OfflineEval}, \mathsf{sid}, I, \mathcal{A}, x, y)$, record $\langle M, \mathcal{A}, I, \bot, (g_j, u)\rangle$, set $H_2(x, u) \coloneqq y, I \coloneqq I + 1$.

Figure 4.13: Reduction to the OMDH problem.

3DH

$\underline{\mathsf{P} \text{ on } \textsc{Init}}$	$\underline{\mathsf{P} \text{ on } Y, B, \mathsf{aux} \text{ from } \mathsf{P}'}$
$a \xleftarrow{\$} \mathbb{Z}_p$, $A := g^a$	retrieve $(\mathsf{sk}, \mathsf{pk}) = (a, A)$ for P
store $(\mathsf{sk}, \mathsf{pk}) = (\mathsf{sid}, a, A)$ for P	if $\mathsf{P} <_{\text{lex}} \mathsf{P}'$
	$Z := B^x \parallel Y^a \parallel Y^x$
$\underline{\mathsf{P} \text{ on } (\textsc{NewSession}, \mathsf{P}')}$	$k := H(\mathsf{aux}, X, Y, Z)$
$x \xleftarrow{\$} \mathbb{Z}_p$, $X := g^x$	else
retrieve $\mathsf{pk} = A$ for P and sid	$Z := Y^a \parallel B^x \parallel Y^x$
send $X, A, \perp$ to P'	$k := H(\mathsf{aux}, Y, X, Z)$

Figure 4.14: Triple Diffie–Hellman Key Exchange 3DH as used in the WBP.

provided with the same auxiliary input. This reflects that in real-world scenarios, two parties only compute the same key if they agree on the context. Second, we introduce the interface Init, which reflects that in 3DH parties generate a long-term secret key when they are initialized.

Theorem 4. *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and $H : \{0,1\}^* \times \mathbb{G}^3 \to \{0,1\}^\lambda$ be a hash function. Suppose the GapCDH Assumption holds in $\mathbb{G}$ and let H be a random oracle.*

Then the 3DH protocol of Figure 4.14 UC-realizes $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ of Figure 4.15. More precisely, for any efficient adversary against 3DH, there is an efficient simulator $\textsc{Sim}_{\mathsf{3DH}}$ that interacts with $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ and produces a view such that for any efficient environment $\mathcal{Z}$ it holds that

$$\mathbf{Adv}^{\mathcal{F}_{\mathsf{AKE\text{-}KCI}}}_{\mathsf{3DH},\textsc{Sim}_{\mathsf{3DH}},\mathcal{Z}}(\lambda) \leq \mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{B}_1}(\lambda) + 2q_I \cdot \mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{B}_2}(\lambda) + \frac{q_S^2}{p},$$

where q_I denotes the number of Init *queries to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ and q_S the number of* NewSession *queries to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$.*

Proof. We describe the simulator $\textsc{Sim}_{\mathsf{3DH}}$ in Figure 4.16. We now show a sequence of hybrid experiments $\mathbf{G}_0, \ldots, \mathbf{G}_6$, where starting from the real-world execution $\textsc{Exec}_{\mathsf{3DH},\mathcal{Z}}$ we make small incremental changes until we reach the ideal-world execution $\textsc{Ideal}_{\mathcal{F}_{\mathsf{AKE\text{-}KCI}}}$ with $\textsc{Sim}_{\mathsf{3DH}}$. While some steps are similar to the proof in [GJK21], we cannot fully adapt their proof due to the different functionalities. We write $\Pr[\mathbf{G}_i]$ as shorthand for the probability that the environment outputs 1 in $\mathbf{G}_i$. Let $q_H \in \mathbb{N}$ denote the number of H queries and $q_I \in \mathbb{N}$ the number of Init queries to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$.

Game $\mathbf{G}_0$: This is the real world execution $\textsc{Exec}_{\mathsf{3DH},\mathcal{Z}}$.

Game $\mathbf{G}_1$: In this game we move everything the protocol parties do to the simulator who internally executes all parties. We also add an ideal functionality that does nothing but forwarding every input it gets to the simulator. To

In the description below, we assume $\mathsf{P}, \mathsf{P}' \in \{\mathsf{U}, \mathsf{S}\}$.

- On $(\textsc{Init}, \mathsf{sid})$ from P, send $(\textsc{Init}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{A}$.
- On $(\textsc{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}')$ from P, send $(\textsc{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}')$ to $\mathcal{A}$. If ssid was not used before by P, record $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}' \rangle$ and mark it $\textsc{Fresh}$.
- On $(\textsc{Compromise}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$, mark P $\textsc{Compromised}$.
- On $(\textsc{Impersonate}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P})$ from $\mathcal{A}$, if P is marked $\textsc{Compromised}$ and there is a record $\langle \mathsf{ssid}, \mathsf{P}, * \rangle$ marked $\textsc{Fresh}$, mark this record $\textsc{Compromised}$.
- On $(\textsc{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{aux}, \mathsf{shk}^*)$ from $\mathcal{A}$, where $|\mathsf{shk}^*| = \lambda$, if there is a record $\langle \mathsf{ssid}, \mathsf{P}, [\mathsf{P}'] \rangle$ not marked $\textsc{Completed}$, do:
  - If the record is marked $\textsc{Compromised}$, or P or P' is corrupted, set $\mathsf{shk} := \mathsf{shk}^*$.
  - If the record is marked $\textsc{Fresh}$, an output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{aux'}, \mathsf{shk}')$ was sent to P' from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ while record $\langle \mathsf{ssid}, \mathsf{P}', \mathsf{P} \rangle$ was marked $\textsc{Fresh}$, and $\mathsf{aux} = \mathsf{aux}'$, set $\mathsf{shk} := \mathsf{shk}'$.
  - Else pick $\mathsf{shk} \xleftarrow{\$} \{0,1\}^\lambda$.

  Finally, mark $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}' \rangle$ $\textsc{Completed}$ and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{aux}, \mathsf{shk})$ to P.

Figure 4.15: Ideal functionality $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. The changes over [JKX18] are marked with gray boxes.

Simulator $\textsc{Sim}_{\mathsf{3DH}}$

1. On $(\textsc{Init}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$: choose $a \xleftarrow{\$} \mathbb{Z}_p$, $A \xleftarrow{\$} g^a$. Store $\langle \mathsf{P}, \mathsf{sid}, a, A \rangle$

2. On $(\textsc{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}')$ from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$:
   - if $\mathsf{P} <_{\text{lex}} \mathsf{P}'$, set $r \coloneqq 0$, else set $r \coloneqq 1$.
   - choose $x \xleftarrow{\$} \mathbb{Z}_p$, set $X = g^x$
   - retrieve $\langle \mathsf{P}, \mathsf{sid}, *, [A] \rangle$
   - store $\langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}', r, x, X \rangle$ and send $X, A, \perp$ to P'.

3. On adversarial message $\mathsf{ssid}, Y, B, \mathsf{aux}$ to P on behalf of P': if there is a record $\langle [\mathsf{sid}], \mathsf{ssid}, \mathsf{P}, \mathsf{P}', [r], [x], * \rangle$:
   - if there is no record $\langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}', \mathsf{P}, *, [y], Y \rangle$ and P' is compromised in session sid, send $(\textsc{Impersonate}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}')$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ and do:
     - if $r = 0$, set $h \coloneqq H(\mathsf{aux}, B^x \parallel Y^a \parallel Y^x)$
     - else, set $h \coloneqq H(\mathsf{aux}, Y^a \parallel B^x \parallel Y^x)$
   - Otherwise, set $h \coloneqq \perp$
   - send $(\textsc{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{aux}, h)$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$.

4. On $(\textsc{Compromise}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{Z}$: Send $(\textsc{Compromise}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, retrieve record $\langle \mathsf{P}, \mathsf{sid}, [a], * \rangle$ and output a to $\mathcal{Z}$.

5. On query (aux, Z) to random oracle H from $\mathcal{Z}$:
   - if there exists a record $\langle H, (\mathsf{aux}, Z), [k] \rangle$, output k.
   - Else pick $k \xleftarrow{\$} \{0,1\}^\lambda$ and store record $\langle H, (\mathsf{aux}, Z), k \rangle$. Output k.

Figure 4.16: Simulator $\textsc{Sim}_{\mathsf{3DH}}$ showing that 3DH UC-realizes $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$.

make the changes oblivious to the environment we also add dummy parties that forward the input they get from $\mathcal{Z}$ to the functionality. Finally, we equip the functionality with dummy interfaces that allow the simulator to let any party produce any output chosen by the simulator. As these are only syntactical changes, we have

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Game $\mathbf{G}_2$: Whenever an honest party P sends a message $X, A, \perp$ for $X = g^x$ such that X was already sent by another honest party $\mathsf{P}' \neq \mathsf{P}$, i.e., the simulator stored a record $\langle *, *, \mathsf{P}', *, *, x, X\rangle$, the experiment aborts. Since $x \xleftarrow{\$} \mathbb{Z}_p$ was sampled uniformly at random and there are at most q_S queries to NewSession, due to the birthday bound we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq \frac{q_S^2}{p}.$$

Game $\mathbf{G}_3$: In this game, we abort if the environment queries H on input aux, Z where $Z = L \,\|\, M \,\|\, N$ corresponds to the output of two honest parties. More precisely, if the simulator stored records $\langle \mathsf{P}, \mathsf{sid}, a, A\rangle, \langle \mathsf{P}', \mathsf{sid}, b, B\rangle, \langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}', 0, x\rangle$, and $\langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}', \mathsf{P}, 1, y\rangle$, s.t. $L = B^x, M = Y^a$, $N = g^{xy}$, we abort the game. Now, we can construct an adversary $\mathcal{B}_1$ that wins the GapCDH game if $\mathcal{Z}$ ever makes a query of this form. The reduction works as follows.

At the start of the game, it receives a CDH challenge $(\bar{X}, \bar{Y})$. Then, for every message $(\text{NewSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}')$ from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, where P is honest, it simulates the message of P by choosing $s \xleftarrow{\$} \mathbb{Z}_p$ and setting $X := \bar{X}^s$ if $r = 0$, or $t \xleftarrow{\$} \mathbb{Z}_p, Y := \bar{Y}^t$ if $r = 1$. Instead of storing $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', *, x, X\rangle$, as $x = \mathsf{dlog}_g(X)$ is not known to the reduction, it stores $\langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}', *, s, X\rangle$, resp. $\langle \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{P}', *, t, Y\rangle$. Now, on a query $\mathsf{aux}, L \,\|\, M \,\|\, N$ to H, the reduction can check with its DDH oracle if $(\bar{X}, Y^s, N)$ or $(\bar{Y}, X^t, N)$ are valid DH triples for any of the recorded pairs s, X and t, Y. If the check is successful, the reduction returns $N^{1/st}$ as result to its challenger. It is easy to see that the reduction wins iff the abort happens. Thus, we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{B}_1}(\lambda).$$

Game $\mathbf{G}_4$: In this game, for honest parties that receive a message from an honest party, we let the functionality compute the session key instead of the simulator. That is, on an adversarial message Y, B, aux from P' to P, the simulator checks if there is a record $\langle [\mathsf{sid}], [\mathsf{ssid}], \mathsf{P}', \mathsf{P}, *, *, Y\rangle$ and a record $\langle \mathsf{P}', \mathsf{sid}, *, B\rangle$. If such records exists, this means that P' is honest and that Y and B were generated by the simulator. It then makes the functionality output a key $\mathsf{shk} \xleftarrow{\$} \{0,1\}^\lambda$ or shk', if the functionality already output a

key shk' to P' in the same subsession ssid of session sid. In $\mathbf{G}_4$, P therefore always outputs a uniformly random key. Conversely, in $\mathbf{G}_3$ the key for P was computed by the simulator as $H(\mathsf{aux}, B^x \parallel Y^a \parallel Y^x)$. However, due to the abort condition introduced in $\mathbf{G}_3$ the environment cannot query the random oracle for this exact input. Therefore, the output of P in $\mathbf{G}_4$ and $\mathbf{G}_3$ is indistinguishable and we have

$$\Pr[\mathbf{G}_4] = \Pr[\mathbf{G}_3].$$

Game $\mathbf{G}_5$: In this game, we change the way honest parties receive their output if the received message Y, B, aux was sent maliciously, i.e., there is no record $\langle [\mathsf{sid}], [\mathsf{ssid}], \mathsf{P}', \mathsf{P}, *, y, Y\rangle$ or $\langle \mathsf{P}', \mathsf{sid}, *, B\rangle$. We distinguish three cases here:

- P' is corrupt: We continue to let the functionality output a key shk^* to P that was provided by the simulator. In $\mathbf{G}_3$ the key of any such session was computed as $H(\mathsf{aux}, B^x \parallel Y^a \parallel Y^x)$ or $H(\mathsf{aux}, Y^a \parallel B^x \parallel Y^x)$, depending on the role of P. In $\mathbf{G}_5$ the simulator gives exactly that value to the functionality.
- P' is compromised: The simulator first sends $(\textsc{Impersonate}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}')$ to $\mathcal{F}_{\mathsf{PPKR}}$ to mark this session as $\textsc{Compromised}$. Afterwards, we continue as in the case above.
- P' is honest and not compromised: In $\mathbf{G}_3$, P outputs a key that was computed as $H(\mathsf{aux}, B^x \parallel Y^a \parallel Y^x)$ or $H(\mathsf{aux}, Y^a \parallel B^x \parallel Y^x)$, depending on its role. In $\mathbf{G}_5$, we let the functionality output a uniformly random key $\mathsf{shk}^* \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{Z}$ can only notice the difference by querying $\mathsf{aux}, B^x \| Y^a \| Y^x$ to H, where B is the public key of P'. If $\mathcal{Z}$ makes such a query, we abort the game. Since P' is honest and not compromised, b s.t. $B = g^b$ is unknown to $\mathcal{Z}$. We can thus create an adversary $\mathcal{B}_2$ that wins the GapCDH game if the environment ever queries $H(\mathsf{aux}, B^x \| Y^a \| Y^x)$, if P has role $r = 0$.
  The reduction works as follows: On a challenge $(\bar{X}, \bar{B})$ the reduction guesses an index $i \in \{1, \dots q_I\}$ and on the i-th $(\textsc{Init}, *, *)$ output from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ it outputs $\bar{B}$. On every $\textsc{NewSession}$ message from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, the reduction chooses $s \xleftarrow{\$} \mathbb{Z}_p$ and computes $X := \bar{X}^s$. It outputs X as message for that party and, similarly as the reduction in game $\mathbf{G}_3$, stores s instead of x. When the reduction receives a query $H(\mathsf{aux}, L \parallel M \parallel N)$ it uses its DDH oracle to check if $(\bar{B}, X, L)$ is a valid DH triple for any recorded X. If that is the case, it retrieves the s stored alongside X and outputs $L^{1/s}$ as to its challenger. It is easy to see that the reduction succeeds if the guessed index i is correct. If P has role $r = 1$ and $\mathcal{Z}$ queries $H(\mathsf{aux}, Y^a \parallel B^x \parallel X^y)$, then we construct an analogous reduction that solves the GapCDH problem in essentially the same way.

Overall we get

$$|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| \leq 2q_I \cdot \mathbf{Adv}^{\mathsf{GapCDH}}_{\mathbb{G},\mathcal{B}_2}(\lambda).$$

Game $\mathbf{G}_6$: In this step we replace the simulator and the functionality described in $\mathbf{G}_5$ with the simulator and the functionality from Figure 4.16 and Figure 4.15. One can verify that this does not change the distribution of the experiment, that is,

$$\Pr[\mathbf{G}_6] = \Pr[\mathbf{G}_5].$$

Combining all probabilities yields the bound claimed in the theorem.

□

4.4.3 Security of the WBP

Modeling the HSM. We model the HSM as a hybrid functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ that can be queried by the server as in Figures 4.4 and 4.5. That is, $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ contains exactly the code that the HSM contains in these figures. For completeness and clarity, $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ is depicted in Figure 4.17. In addition, $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ provides an interface for clients to retrieve the HSM's public key, which models the setup process that ensures that clients have the "right" WhatsApp public key hard-coded into their smartphones.[18]

In the UC framework, messages sent between some party and an ideal functionality are perfectly secure, meaning no network adversary can intercept them or tamper with them. Thus, modeling the HSM key distribution as idealized communication expresses our assumptions that

1. the user installs the correct WhatsApp client on her phone,
2. WhatsApp's setup ceremony of the HSM leads to honestly generated keys being distributed to the clients, and
3. only the HSM knows the secret key of the HSM.

However, analyzing the mechanisms to ensure the above assumptions is not in the scope of this work.

Corruption Model. All corruptions are malicious, meaning that the adversary can fully control not only the communication but also the behavior of a corrupted party. We consider adaptive corruptions of clients and the server, however, we add the restriction that clients cannot be corrupted during an ongoing initialization or retrieval session. More precisely, the environment is not allowed to corrupt some client cid if, following the most recent $(\textsc{InitC}, *)$ input from the environment to cid, cid did not produce a corresponding output $(\textsc{InitRes}, *)$ yet. Analogously, if following the most recent $(\textsc{RetC}, *)$ input from the environment to cid, cid did not produce a corresponding output $(\textsc{RetRes}, *)$ yet, the environment is not

[18] One might be tempted to model this by giving the HSM's public key as input to the client instead. However, that would mean that the UC environment machine can give public keys to clients for which the environment knows the corresponding secret key. For WBP the clients have a hard-coded public key for which only the HSM knows the secret key, so this would not adequately model WBP and make the already complex security analysis unreasonably more complex.

Ideal functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$

Initially:
- $\mathsf{aids} \coloneqq \emptyset$
- $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{Sig.KGen}(1^\lambda)$
- $(\mathsf{sk}_{\mathsf{Enc}}, \mathsf{pk}_{\mathsf{Enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$
- $y \xleftarrow{\$} \mathbb{Z}_p, Y \coloneqq g^y$
- Set $\mathsf{pk} \coloneqq \{\mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Enc}}, Y\}$ and $\mathsf{sk} \coloneqq \{\mathsf{K}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Enc}}, y\}$

On (GetPK) from $\mathsf{P} \in \{\mathsf{cid}, \mathsf{S}, \mathcal{A}\}$:
- Send pk to P

On (InitS, $\mathsf{aid}, \mathsf{aid}', a_1$) from S:
- If $\mathsf{aid} \in \mathsf{aids}$, return (InitRes, aid, Fail)
- Delete $\langle \mathsf{aid}', *, *, * \rangle$
- $\mathsf{aids} \coloneqq \mathsf{aids} \cup \{\mathsf{aid}\}$
- $\mathsf{k}_{\mathsf{OPRF}} \xleftarrow{\$} \mathbb{Z}_p$
- $b_1 \coloneqq a_1^{\mathsf{k}_{\mathsf{OPRF}}}$
- $n_1 \xleftarrow{\$} \{0,1\}^\lambda$
- $\sigma_1 \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{Sig}}, b_1 \parallel n_1)$
- $\mathsf{tr}' \coloneqq \mathsf{H}_3(a_1, b_1, n_1)$
- Send $(\mathsf{aid}, b_1, n_1, \sigma_1)$ to S

On (File, aid, E) from S:
- $(\mathsf{e} \parallel \mathsf{tr} \parallel X \parallel \mathsf{e\_cred} \parallel n_e \parallel T_e) \coloneqq \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{Enc}}, \mathsf{E})$
- If $\mathsf{tr} \neq \mathsf{tr}'$, send (InitRes, aid, Fail) to S
- If $\mathsf{tr} = \mathsf{tr}'$, store $\langle \mathsf{aid}, \mathsf{k}_{\mathsf{OPRF}}, (\mathsf{e}, X, \mathsf{e\_cred}, n_e, T_e), 10 \rangle$ and send (InitRes, aid, Succ) to S

On (RetS, $\mathsf{aid}, n_C, U, a_2$) from S:
- retrieve $\langle \mathsf{aid}, [\mathsf{k}_{\mathsf{OPRF}}], [m], [\mathsf{ctr}] \rangle$
- $\mathsf{e}, X, \mathsf{e\_cred}, n_e, T_e \coloneqq m$
- If no record can be found, send (RetRes, aid, Fail) to S, else continue
- If $\mathsf{ctr} = 0$, delete $\langle \mathsf{aid}, *, *, * \rangle$ and send (DelRec, aid) to S, else continue
- Update $\langle \mathsf{aid}, *, *, * \rangle$ with $\mathsf{ctr} - 1$
- $b_2 \coloneqq a_2^{\mathsf{k}_{\mathsf{OPRF}}}$
- $n_S \xleftarrow{\$} \{0,1\}^\lambda$
- $v \xleftarrow{\$} \mathbb{Z}_p, V \coloneqq g^v$
- $\mathsf{pre} \coloneqq (a_2, n_c, U, Y, \mathsf{e\_cred}, n_e, T_e, b_2, n_S, V)$
- $\mathsf{ikm} \coloneqq (U^v, U^y, X^v)$
- $(\mathsf{K}_S^{\mathsf{MAC}}, \mathsf{K}_C^{\mathsf{MAC}}, \mathsf{shk}) \coloneqq \mathsf{KDF}_2(\mathsf{ikm}, \mathsf{pre})$
- $T_S \coloneqq \mathsf{MAC.Tag}(\mathsf{K}_S^{\mathsf{MAC}}, \mathsf{H}_3(\mathsf{pre}))$
- $\sigma_2 \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{Sig}}, b_2)$
- Send $(\mathsf{aid}, b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2)$ to S

On (RetRes, aid, T_C) from S
- If $\mathsf{MAC.Vrfy}(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{H}_3(\mathsf{pre} \parallel T_S), T_C) = 0$, send (RetRes, aid, Fail) to S, else continue
- $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, \mathsf{e})$
- Update $\langle \mathsf{aid}, *, *, * \rangle$ with $\mathsf{ctr} \coloneqq 10$
- Send (aid, c) to S

Figure 4.17: The ideal functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$.

allowed to corrupt cid as well. We introduce this restriction due to the following reasoning.

Imagine that the environment instructs some honest client cid to first initialize with some password pw and afterwards instructs cid to start a recovery phase, again using pw. Then, after the server produced its final message c, but before the environment allows c to be delivered to cid, it instructs the adversary to corrupt cid. It now learns the entire state of cid, and in particular the keys K_{export} and shk that cid would use to obtain K. Thus, it can now check whether the ciphertext c was produced according to the protocol description, namely by computing $\mathsf{e} \coloneqq \mathsf{AE.Dec}(\mathsf{shk}; c)$, $K \coloneqq \mathsf{AE.Dec}(K_{\mathsf{export}}; \mathsf{e})$, and checking whether $K \stackrel{?}{=} K'$, where K' is the key that was output by cid in the initialization phase.

However, in the ideal world, the simulator could not produce c according to the protocol description, as it in particular does not know K'. Instead it can only output a simulated ciphertext $c' \coloneqq \mathsf{AE.Enc}(\mathsf{shk}, 0^\lambda)$. Therefore, upon corrupting cid the environment would be able to efficiently distinguish between the real and ideal world. In order to avoid this, we disallow the corruption of clients during ongoing initialization or recovery sessions, which prevents the environment from learning shk and checking whether c indeed contains e and in turn K' since cid deletes shk before outputting K at the end of the recovery phase.

We believe that even with the added restriction, this still provides a reasonably realistic modeling of corruptions. In practice we expect a full intialization or recovery session to take only a few seconds and thus we expect it to be very difficult for an adversary to corrupt a client in this short timeframe. Note that an adversary could theoretically extend this "corruption timeframe" by e.g. recording c and dropping the message from the network. However, we assume that in practice all protocol participants implement some timeout mechanism, which terminates the session if no response arrived within a short timeframe, as is standard in any networked application.

Formally, this means that the effect of adaptive client corruptions is that the adversary (1) learns all values that the client stores after completion of an initialization or retrieval phase, namely key K, and (2) controls the client behavior from that point on.

Let $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be the AEAD scheme that is implicitly used in Figures 4.4 and 4.5 to encrypt x, that is:

$\mathsf{KGen}(1^\lambda)$**:** Outputs $(K_{\mathsf{mask}}, K_{\mathsf{auth}})$, where $K_{\mathsf{mask}}, K_{\mathsf{auth}} \xleftarrow{\$} \{0,1\}^\lambda$

$\mathsf{Enc}((K_{\mathsf{mask}}, K_{\mathsf{auth}}), x, Y \parallel n_e)$**:** Computes $\mathsf{e\_cred} \coloneqq x \oplus K_{\mathsf{mask}}$ and $T_e \xleftarrow{\$} \mathsf{MAC.Tag}(K_{\mathsf{auth}}, Y \parallel n_e \parallel \mathsf{e\_cred})$ and outputs $(\mathsf{e\_cred}, T_e)$

$\mathsf{Dec}((K_{\mathsf{mask}}, K_{\mathsf{auth}}), (\mathsf{e\_cred}, T_e), Y \parallel n_e)$**:** If $\mathsf{MAC.Vrfy}(K_{\mathsf{auth}}, \mathsf{e\_cred} \| Y \| n_e, T_e) = 0$, outputs $\perp$ and $x \coloneqq \mathsf{e\_cred} \oplus K_{\mathsf{mask}}$ otherwise

For the security analysis, we require AEAD to provide *random-key robustness* [JKX18], which means that given two randomly sampled keys, it should be difficult

to compute a ciphertext that decrypts successfully under both keys. We give the formal definition below.

Definition 34. The advantage of an adversary $\mathcal{A}$ against the *random-key robustness* (RKR) of an AEAD scheme $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{RKR}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \coloneqq \Pr[\mathsf{Dec}(k,c) \neq \bot \wedge \mathsf{Dec}(k',c) \neq \bot \mid k, k' \xleftarrow{\$} \mathcal{K}, c \xleftarrow{\$} \mathcal{A}(k,k')].$$

We say AEAD is RKR-secure if $\mathbf{Adv}^{\mathsf{RKR}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ is negligible in λ for all efficient adversaries $\mathcal{A}$.

We are now ready to state our main theorem of this chapter.

Theorem 5. *Let* $\mathsf{H}_1, \mathsf{H}_2, \mathsf{KDF}_1, \mathsf{KDF}_2$ *be random oracles such that* 2HDH *UC-realizes the "multi-key" OPRF functionality* $\mathcal{F}_{\mathsf{OPRF}}$ *of Figure 4.10 and* 3DH *UC-realizes the AKE functionality* $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ *of Figure 4.15. Let* $\mathsf{Sig} = (\mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Vrfy})$ *be an* sEUF-CMA*-secure signature scheme,* $\mathsf{MAC} = (\mathsf{MAC.KGen}, \mathsf{MAC.Tag}, \mathsf{MAC.Vrfy})$ *be an* sEUF-CMA*-secure MAC,* $\mathsf{PKE} = (\mathsf{PKE.KGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an* IND-CCA*-secure public key encryption scheme,* $\mathsf{AE} = (\mathsf{AE.KGen}, \mathsf{AE.Enc}, \mathsf{AE.Dec})$ *be an authenticated encryption scheme,* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *have random-key robustness, and* $\mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^\lambda$ *be a collision resistant hash function.*

The WBP *as described in Figure 4.4 and Figure 4.5 UC-realizes the functionality* $\mathcal{F}_{\mathsf{PPKR}}$ *of Figures 4.7 to 4.9 with* Lev-1 *security (i.e., without the* LeakFile *and* FullyCorrupt *interface) in the* $\mathcal{F}^{\mathsf{WBP}}_{\mathsf{HSM}}$*-hybrid model, assuming adaptive malicious corruption of clients and the server restricted as described above, and a client-authenticated channel between clients and the server. Concretely, we can construct adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_8$ *and environments* $\mathcal{Z}_1$ *and* $\mathcal{Z}_2$ *such that for any efficient adversary against* WBP *(interacting with* $\mathcal{F}^{\mathsf{WBP}}_{\mathsf{HSM}}$*), the simulator* $\textsc{Sim}_{\mathsf{WBP}}$ *that interacts with* $\mathcal{F}_{\mathsf{PPKR}}$ *produces a view such that for every efficient environment* $\mathcal{Z}$*, it holds that*

$$\begin{aligned}
\mathbf{Adv}^{\mathcal{F}_{\mathsf{PPKR}}}_{\mathsf{WBP},\textsc{Sim}_{\mathsf{WBP}},\mathcal{Z}}(\lambda) \leq\ & \mathbf{Adv}^{\mathcal{F}_{\mathsf{OPRF}}}_{\mathsf{2HDH},\textsc{Sim}_{\mathsf{2HDH}},\mathcal{Z}_1}(\lambda) + \mathbf{Adv}^{\mathcal{F}_{\mathsf{AKE\text{-}KCI}}}_{\mathsf{3DH},\textsc{Sim}_{\mathsf{3DH}},\mathcal{Z}_2}(\lambda) \\
& + \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\Sigma,\mathcal{B}}(\lambda) + 2q_{\textsc{Ret}}\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{B}}(\lambda) \\
& + 2\mathbf{Adv}^{\mathsf{CollRes}}_{\mathsf{H}_3,\mathcal{B}}(\lambda) + q_{\textsc{Init}}\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{B}}(\lambda) \\
& + q_{\textsc{Ret}}\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AE},\mathcal{B}}(\lambda) + q_{\textsc{Ret}}\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AE},\mathcal{B}}(\lambda) \\
& + \binom{q_{\textsc{Init}}}{2}2^{-\lambda} + \binom{q_{\textsc{Ret}}}{2}2^{-\lambda+1} + \binom{q_{\mathsf{KDF}_1}}{2}\mathbf{Adv}^{\mathsf{RKR}}_{\mathsf{AEAD},\mathcal{B}}(\lambda),
\end{aligned}$$

where $q_{\textsc{Init}} \in \mathbb{N}$ *is an upper bound on the number of initializations,* $q_{\textsc{Ret}} \in \mathbb{N}$ *is an upper bound on the number of recoveries,* q_{KDF_1} *is an upper bound on the number of* KDF_1 *queries.*

Proof Intuition. Let us begin with a high-level description of the proof. We have to show that we can simulate protocol transcripts without knowledge of the passwords chosen for honest clients by the environment such that the simulated transcripts are indistinguishable from the real protocol transcripts.

For this, we heavily rely on the security of the 2HDH OPRF and the 3DH AKE, which we have proven in Sections 4.4.1 and 4.4.2. The simulator $\text{SIM}_{\mathsf{WBP}}$ internally runs a copy of the simulators $\text{SIM}_{\mathsf{2HDH}}$ and $\text{SIM}_{\mathsf{3DH}}$ and the corresponding functionalities $\mathcal{F}_{\mathsf{OPRF}}$ and $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ that the simulators interact with. Then, whenever we have to simulate (parts of) a message in the WBP that is part of 2HDH or 3DH for some honest party, we "outsource" the simulation to $\text{SIM}_{\mathsf{2HDH}}$ or $\text{SIM}_{\mathsf{3DH}}$.

Another crucial proof strategy is simulating any ciphertext by encrypting a 0-string whenever the environment cannot know the key used for decryption. In this way we can simulate ciphertexts, which in the real protocol depend on the secret input of some party, without knowledge of the secret input. Due to the IND-CPA or IND-CCA security of the encryption scheme, no environment is able to recognize this change.

Further, we have to show that the outputs parties produce in the simulated execution are indistinguishable from the outputs in the real execution. As long as both parties are honest, this is rather straightforward as we can rely on $\mathcal{F}_{\mathsf{PPKR}}$, which in particular lets us learn whether the client used the correct password for retrievals and provides the output to the parties accordingly. A more difficult challenge arises when some party is corrupt. In that case, we use $\text{SIM}_{\mathsf{2HDH}}$ to extract the password the corrupt party used from the random oracle queries to H_2. We can then provide the extracted password to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of the corrupt party and let $\mathcal{F}_{\mathsf{PPKR}}$ produce the output for the honest party.

Proof. We depict our simulator $\text{SIM}_{\mathsf{WBP}}$ in Figures 4.18 to 4.23. Throughout the proof, we regularly reference steps of the simulator in the form I.1. The simulator works with session state records

$$\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, a, \mathsf{kid}, \mathsf{aid}, a^*, b_1, n_1, \sigma_1, b_1^*, n_1^*, x, \mathsf{E} \rangle$$

for initialization. All values are initialized to $\perp$ and potentially updated throughout an initialization. At the end of an initialization by cid, the record has the following semantics.

INIT_2 = ssid indicates the sub-session id used for this initialization;

INIT_3 $\in \{\mathsf{cid}, \perp\}$ is either the cid of the client running the initialization or it is set to $\perp$ if the corrupt S maliciously initializes;

INIT_4 $\in \{a, \perp\}$ is either the a value sent by an honest cid, or $\perp$ if cid is corrupt or if the corrupt S impersonates cid;

INIT_5 = kid is the OPRF key identifier chosen for this initialization;

$\textsc{Init}_6 \in \{\mathsf{aid}, \perp\}$ is either $\perp$ if the server is honest, or the new account name delivered to the HSM if the server is corrupt;

$\textsc{Init}_7 = a^*$ indicates the a value delivered to the HSM or the server;

$\textsc{Init}_8 = b_1$ indicates the b_1 value sent by the HSM or the server;

$\textsc{Init}_9 = n_1$ indicates the n_1 value sent by the HSM or the server;

$\textsc{Init}_{10} = \sigma_1$ indicates the σ_1 value sent by the HSM or the server;

$\textsc{Init}_{11} \in \{b_1^*, \perp\}$ indicates the b_1 value received by the client or $\perp$ if cid is corrupt;

$\textsc{Init}_{12} \in \{n_1^*, \perp\}$ indicates the n_1 value received by the client or $\perp$ if cid is corrupt;

$\textsc{Init}_{13} \in \{x, \perp\}$ is the client's Diffie–Hellman secret either simulated for an honest client or decrypted from E and $\mathsf{e\_cred}$. It is set to $\perp$ if it cannot be extracted from a corrupted initialization;

$\textsc{Init}_{14} \in \{\mathsf{E}, \perp\}$ is the ciphertext E computed by cid or $\perp$ if cid is corrupt.

Similarly, the simulator works with session state records

$$\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, m_1, match, \mathsf{kid}, \mathsf{aid}, m_1^*, (\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk}), m_2, T_C, T_C^*, c \rangle$$

for retrieval. At the end of a retrieval by cid, the record has the following semantics.

$\textsc{Ret}_2 = \mathsf{ssid}$ indicates the sub-session id used for this initialization;

$\textsc{Ret}_3 \in \{\mathsf{cid}, \perp\}$ is either the cid of the client running the retrieval or it is set to $\perp$ if the corrupt S maliciously retrieves;

$\textsc{Ret}_4 \in \{(n_C, U, a_2), \perp\}$ is either the message (n_C, U, a_2) sent by an honest cid, or $\perp$ if cid is corrupt or if the corrupt S impersonates cid;

$\textsc{Ret}_5 \in \{0, 1\}$ indicates whether the password used in the retrieval is correct

$\textsc{Ret}_6 = \mathsf{kid}$ is the key identifier used in this retrieval;

$\textsc{Ret}_7 \in \{\mathsf{aid}, \perp\}$ is either $\perp$ if the server is honest, or the account name delivered to the HSM if the server is corrupt;

$\textsc{Ret}_8 = (n_C, U, a_2)$ indicates the a value delivered to the HSM or the server;

$\textsc{Ret}_9 = (\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ indicates the keys computed by HSM or the server

$\textsc{Ret}_{10} = (b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2)$ indicates the message sent by the HSM or the server;

$\text{RET}_{11} \in \{T_C, \perp\}$ is either the T_C value sent by an honest cid or $\perp$ if cid is corrupt or if the corrupt S impersonates cid;

$\text{RET}_{12} = T_C^*$ is the T_C value delivered to the HSM or the server;

$\text{RET}_{13} = c$ indicates the ciphertext c sent by the HSM or the server

We construct a sequence of games $\mathbf{G}_0$ to $\mathbf{G}_{23}$, where we gradually change the real-world execution of the protocol WBP to reach the ideal-world execution, where the environment interacts with the simulator from Figures 4.18 to 4.23 and the ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$. We write $\Pr[\mathbf{G}_i]$ to denote the probability that the environment outputs 1 in the game $\mathbf{G}_i$.

Game $\mathbf{G}_0$: Real world. This is the real world.

Game $\mathbf{G}_1$: Create simulator. In this game we create two new entities: an ideal functionality $\mathcal{F}$ and a simulator Sim. The functionality $\mathcal{F}$ forwards all inputs it receives to Sim and provides an interface to Sim that allows Sim to let $\mathcal{F}$ produce any given output to any given party. The simulator Sim executes the protocol code of WBP on the input given by $\mathcal{F}$ and provides back the output through $\mathcal{F}$. Additionally, Sim stores the session state records described above just like $\text{Sim}_{\mathsf{WBP}}$. To make the introduction of $\mathcal{F}$ oblivious to the environment, we also add dummy parties that just forward any input they receive from the environment to $\mathcal{F}$, resp. from $\mathcal{F}$ to the environment. Finally, we equip $\mathcal{F}$ with the interfaces Corrupt, LeakFile, MaliciousInit, MaliciousRet, and OfflineAttack with the exact same code as in $\mathcal{F}_{\mathsf{PPKR}}$.

This are merely syntactical changes as still every message is produced as in the real-world. Thus, this game is identically distributed as $\mathbf{G}_0$, i.e.,

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Game $\mathbf{G}_2$: Do not maintain aid while S is honest. In this game the simulator does not maintain aids while S is honest and only samples an aid for each cid when S gets corrupted (see C.1). Furthermore, after the server gets corrupted, Sim uses the map $\mathsf{aids}[\cdot]$ to store the aid of the most recent initialization of each cid (see C.1, Ia.12, IE.5, IE.6 (d), R.1, and Ra.3). This is again a purely syntactical change that we do for bookkeeping in the simulator. Since the honest server maintains a one-to-one mapping and all messages sent between cid and S and the outputs of the parties are independent of aid, this change is oblivious to the environment. Hence, we have

$$\Pr[\mathbf{G}_2] = \Pr[\mathbf{G}_1].$$

Game $\mathbf{G}_3$: Abort upon H_2 collision. In this game the simulator aborts if the random oracle H_2 ever outputs the same value ρ for two different inputs (H2.3). Since the outputs are sampled uniformly at random, due to the birthday bound we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \frac{q_{\mathsf{H}_2}^2}{2^\lambda},$$

where q_{H_2} is the number of queries to H_2.

Game $\mathbf{G}_4$: Abort upon K_{auth} collision. In this game the simulator aborts if the random oracle KDF_1 ever outputs the same value K_{auth} for two different inputs (K1.1 (c)). Since the outputs are sampled uniformly at random, due to the birthday bound we have

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \frac{q_{\mathsf{KDF}_1}^2}{2^\lambda},$$

where q_{KDF_1} is the number of queries to KDF_1.

Game $\mathbf{G}_5$: Abort upon nonce collision. In this game we let SIM abort if there is a collision on the nonces n_1 sampled in two different initializations (Ia.7). Similarly, we also abort on collisions on n_e, n_C, n_S (Ib.8 , R.6 , Ra.5). Since SIM samples a nonce n_1 and n_e in each initialization and a nonce n_C and n_S in each retrieval, due to the birthday bound we have

$$|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| \leq \frac{2(q_{\text{INIT}}^2 + q_{\text{RET}}^2)}{2^\lambda},$$

where q_{INIT} is the number of initializations and q_{RET} is the number of retrievals.

Game $\mathbf{G}_6$: Output FAIL upon adversarial signature. In this game we let SIM immediately output FAIL to any cid if some σ_1 or σ_2 that was not computed by SIM is delivered to cid (see Ib.3 and Rb.3). Note that as in $\mathbf{G}_5$ the adversary can still replay some message b_1, n_1, σ_1, which is reflected in Ib.3 by not specifying an ssid for the INIT record. $\mathbf{G}_6$ and $\mathbf{G}_5$ only deviate if a valid signature is sent to cid that was not issued by the SIM before, which we denote as the event $\mathrm{E}_{\mathsf{Sig}}$. If this happens with non-negligible probability, we can construct an adversary against the $\mathsf{sEUF\text{-}CMA}$ security of Sig.

The reduction works as follows: assume there is an environment $\mathcal{Z}^*$ that causes the event $\mathrm{E}_{\mathsf{Sig}}$ when interacting with $\mathcal{F}$ and SIM. Then, the reduction $\mathcal{B}_1$ internally runs the whole experiment including $\mathcal{F}$, SIM and $\mathcal{Z}^*$. Instead of letting the simulator compute $\mathsf{pk}_{\mathsf{Sig}}$ itself, $\mathcal{B}_1$ uses the public key that it gets from the challenger. Moreover, instead of computing σ_1 and σ_2 by itself, $\mathcal{B}_1$ asks the oracle SIGN to produce the signature. When the simulator detects $\mathrm{E}_{\mathsf{Sig}}$, $\mathcal{B}_1$ outputs the signature σ^* and the message m^* that caused

$\mathrm{E}_{\mathsf{Sig}}$ to its challenger. It is easy to see that $\mathcal{B}_1$ wins the sEUF-CMA game whenever $\mathrm{E}_{\mathsf{Sig}}$ occurs. Thus, we have

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda).$$

Game $\mathbf{G}_7$: cid encrypts 0 in E. In this game we change the computation of the ciphertext E. Whenever Sim simulates E for some honest cid, it computes $\mathsf{E} \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{Enc}}, m)$, where m is a 0-string of appropriate length. Furthermore, it stores a record $\langle \mathsf{E}, X \,\|\, \mathsf{tr} \,\|\, \mathsf{e\_cred} \,\|\, n_e \,\|\, T_e \rangle$ that stores the values that would normally be encrypted in E and retrieves the values from this record when E is delivered to S (see Ib.7, IE.3 (b), and IE.5).[19]

Assume there is an environment $\mathcal{Z}^*$ that can distinguish $\mathbf{G}_7$ and $\mathbf{G}_6$. We construct an adversary $\mathcal{B}_2$ against the IND-CCA security of PKE as follows. First, $\mathcal{B}_2$ sets $\mathsf{pk}_{\mathsf{Enc}} := \mathsf{pk}$, where pk is the public key received from its challenger. Then, we construct a sequence of games $\mathbf{G}_6^{(0)} := \mathbf{G}_6, \mathbf{G}_6^{(1)}, \ldots, \mathbf{G}_6^{(q_{\text{Init}})} := \mathbf{G}_7$, where in $\mathbf{G}_6^{(i)}$ in the first i initializations E is computed as an encryption of a 0-string and in the remaining initializations it is computed as in $\mathbf{G}_6$. Because $\mathcal{Z}^*$ can distinguish between $\mathbf{G}_7$ and $\mathbf{G}_6$, there must be an index $i^* \in \{1, \ldots, q_{\text{Init}}\}$ such that $\mathcal{Z}^*$ can distinguish between $\mathbf{G}_7^{(i^*-1)}$ and $\mathbf{G}_7^{(i^*)}$. Now, in the i^*-th initialization, $\mathcal{B}_2$ outputs $m_0 := X \,\|\, \mathsf{tr} \,\|\, \mathsf{e\_cred} \,\|\, n_e \,\|\, T_e$ and m_1 as a 0-string of the same length to its challenger to receive a challenge ciphertext c^* and sets $\mathsf{E} := c^*$. Additionally, whenever it receives some message $(\mathsf{ssid}, \mathsf{E})$ to S or $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ such that no record $\langle \mathsf{E}, * \rangle$ exists, it uses its decryption oracle Dec to decrypt E. It is easy to see that if the challenger encrypts m_0, the game is distributed exactly like $\mathbf{G}_6^{(i^*-1)}$ and if it encrypts m_1, it is distributed exactly like $\mathbf{G}_6^{(i^*)}$. Hence, we get

$$|\Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6]| \leq q_{\text{Init}} \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_2}(\lambda).$$

Game $\mathbf{G}_8$: Abort on collision in H_3. In this game, we let Sim abort if it ever computes the same output of H_3 for two different inputs. It is easy to see that we can construct an adversary $\mathcal{B}_3$ against the collision resistance of H_3 if any environment $\mathcal{Z}^*$ causes Sim to abort with non-negligible probability. $\mathcal{B}_3$ runs the whole experiment with $\mathcal{Z}^*$ and outputs the two inputs that led to the abort. We therefore have

$$|\Pr[\mathbf{G}_8] - \Pr[\mathbf{G}_7]| \leq \mathbf{Adv}^{\mathsf{CollRes}}_{\mathsf{H}_3,\mathcal{B}_3}(\lambda).$$

Game $\mathbf{G}_9$: Skip tr validation. In this game, the simulator does not check the transcript tr' anymore whenever cid is honest and either S is honest or the corrupt S honestly delivers E. Instead it simply checks whether the values

[19] Due to the changes later introduced in $\mathbf{G}_9$, we do not store tr in the record in Ib.7 .

a_1, b_1, and n_1 were honestly delivered by $\mathcal{A}$ and outputs Fail to S if any was not honestly delivered (IE.3 (a)).

Let a_1, b_1, n_1 denote the values simulated by Sim and a_1^*, b_1^*, n_1^* denote the values delivered by $\mathcal{A}$. $\mathbf{G}_9$ and $\mathbf{G}_8$ are clearly identical until $\mathcal{A}$ delivers some a_1^*, b_1^*, n_1^* such that $\mathsf{H}_3(a_1, b_1, n_1) = \mathsf{H}_3(a_1^*, b_1^*, n_1^*)$. However, due to the changes introduced in $\mathbf{G}_8$, Sim aborts the simulation if that ever happens. We therefore have

$$\Pr[\mathbf{G}_9] = \Pr[\mathbf{G}_8].$$

Game $\mathbf{G}_{10}$: Replace the 2HashDH protocol. In this game we change the simulators behavior concerning the messages of the 2HashDH protocol. Instead of computing the messages of the 2HashDH protocol as in the real-world, the simulator uses its internal ideal OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ and its internal OPRF simulator $\text{Sim}_{\mathsf{2HDH}}$ to simulate the messages and outputs. In particular, this affects the messages a_1 (cf. I.2), a_2 (cf. R.2), b_1 (cf. Ia.6 and Ia.13), and b_2 (cf. Ra.3 and Ra.9), the computation of ρ (cf. Ib.4) and queries to H_1 (cf. H1.1) and H_2 (cf. H2.2 and H2.4). Moreover, we ensure that $\text{Sim}_{\mathsf{2HDH}}$ simulates a fresh OPRF key (cf. Ia.5) in each initialization by choosing a fresh kid (cf. I.1 , Ia.2 (a), and Ia.9 (a)). Finally, instead of storing $\mathsf{k}_{\mathsf{OPRF}}$ in the File records, we now store kid (cf. IE.2 (b), IE.3 (b), IE.6 (e)) and retrieve kid in retrievals before simulating b_2 (cf. R.1 , Ra.2 , and Ra.7).

One challenge in this game comes from the fact that the adversary can replay messages $(\hat{a}_1, \hat{n}_1, \hat{\sigma}_1)$. A client cannot recognize that the message is replayed and just proceeds normally, however with overwhelming probability Sim will output Fail to the server due to $\mathbf{G}_9$. Since $\text{Sim}_{\mathsf{2HDH}}$ simulated $\hat{b}_1$ as $\hat{a}_1^{\hat{k}}$ for a uniformly random key $\hat{k}$ (see Steps 2 and 6 of $\text{Sim}_{\mathsf{2HDH}}$), $\hat{b}_1$ is a uniformly random element from $\mathbb{G}$. Since the same holds for the message b_1 that Sim simulates when it receives the message a_1 from cid, the probability that the check in IE.3 (a) succeeds when cid received a replayed message is at most $1/q$, where q is the order of $\mathbb{G}$. For this reason we can simply sample ρ uniformly at random when cid receives a replayed message (see Ib.5) without interacting with $\text{Sim}_{\mathsf{2HDH}}$.

Now, assume there is an environment $\mathcal{Z}^*$ that can distinguish between games $\mathbf{G}_9$ and $\mathbf{G}_{10}$. We can construct an environment $\mathcal{Z}_1$ that can distinguish between the real 2HashDH protocol and the simulated execution with $\text{Sim}_{\mathsf{2HDH}}$ and $\mathcal{F}_{\mathsf{OPRF}}$. The environment $\mathcal{Z}_1$ internally runs the whole experiment including $\mathcal{F}$ and Sim with $\mathcal{Z}^*$, but whenever Sim would execute a operation of 2HashDH, it instead generates the corresponding output from the OPRF experiment. If $\mathcal{Z}_1$ interacts with $\mathsf{2HDH}$, it perfectly simulates $\mathbf{G}_9$ for $\mathcal{Z}^*$, and if it interacts with $\mathcal{F}_{\mathsf{OPRF}}$ and $\text{Sim}_{\mathsf{2HDH}}$, it simulates $\mathbf{G}_{10}$ for $\mathcal{Z}^*$ with probability $1 - 1/q$.

It follows that we have

$$|\Pr[\mathbf{G}_{10}] - \Pr[\mathbf{G}_9]| \leq \mathbf{Adv}^{\mathcal{F}_{\mathsf{OPRF}}}_{\mathsf{2HDH},\textsc{Sim}_{\mathsf{2HDH}},\mathcal{Z}_1}(\lambda) - \frac{1}{q}.$$

Game $\mathbf{G}_{11}$: Rule out ambiguous OPAQUE envelopes $\mathsf{e\_cred}, T_e$. In this game, the simulator aborts the simulation if it ever sees two distinct random oracle queries (ρ, n_e) and (ρ', n_e') to KDF_1 such that any $(\mathsf{e\_cred}, T_e, (Y, n_e))$ it has ever computed or received decrypts successfully with AEAD under the output of both queries.

Assume there is an environment $\mathcal{Z}^*$ that can distinguish between games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$ with non-negligible probability. We can construct an adversary $\mathcal{B}_4$ against the RKR-security of AEAD as follows. At the beginning, $\mathcal{B}_4$ receives two keys k_1 and k_2 from its challenger. Additionally, it guesses two random indices $i, j \in \{1, \ldots, Q_{\mathsf{KDF}_1}\}$, where q_{KDF_1} is an upper bound on the number of random oracle queries to KDF_1. $\mathcal{B}_4$ internally runs the whole experiment with $\mathcal{F}$, Sim, and $\mathcal{Z}^*$. On the i-th query to KDF_1, it programs the output to be $K_{\mathsf{export}} \parallel k_1$ for some uniformly random K_{export}, and on the j-th query it programs the output to be $K_{\mathsf{export}}{}' \parallel k_2$ for some uniformly random $K_{\mathsf{export}}{}'$. When $\mathcal{Z}^*$ causes Sim to abort, $\mathcal{B}_4$ outputs the values $(\mathsf{e\_cred}, T_e, (Y, n_e))$ that led to the abort to its challenger. If the the outputs of the queries i and j are the ones under which $(\mathsf{e\_cred}, T_e, (Y, n_e))$ decrypts successfully, then $\mathcal{B}_4$ wins the RKR experiment. Hence, we have

$$|\Pr[\mathbf{G}_{11}] - \Pr[\mathbf{G}_{10}]| \leq \frac{1}{q^2_{\mathsf{KDF}_1}} \mathbf{Adv}^{\mathsf{RKR}}_{\mathsf{AEAD},\mathcal{B}_4}(\lambda).$$

Game $\mathbf{G}_{12}$: Extract pw from malicious initializations. In this game we change the behavior of Sim whenever it receives a message $(\mathsf{ssid}, \mathsf{E})$ that was not computed by Sim on behalf of some honest cid. Here we want to extract the password that the corrupt cid or corrupt S used in this initialization. To this end, let $(\mathsf{e}, \mathsf{tr}, X, \mathsf{e\_cred}, n_e, T_e)$ be the values Sim obtains from decrypting E. We then search for a previous query (pw, h) to the random oracle H_2, such that T_e verifies under the key K_{auth} derived from ρ and n_e, where ρ is the output of the query to H_2. Note that due to $\mathbf{G}_{11}$ there is at most one K_{auth} under which T_e verifies and due to game $\mathbf{G}_4$ there can then only be one pair (ρ, n_e) such that $(*, K_{\mathsf{auth}}, *) = \mathsf{KDF}_1(\rho, n_e)$. Thus, Sim can uniquely determine pw.

If Sim is able to find such a query and thus extract the password pw, it lets $\mathcal{F}$ create a record for cid by sending either InitC and CompleteInitS or MaliciousInit to $\mathcal{F}$. Moreover, it can now decrypt $\mathsf{e\_cred}$ to obtain the DH secret x and store it in the Init record. If it cannot extract a password, it lets $\mathcal{F}$ create the record anyway, however with $\mathsf{pw} = \perp$ since the corrupt party must have computed T_e under some randomly chosen key that was not

output by KDF_1. The distribution of this game does not change as $\mathcal{F}$ only creates the FILE records but does not use them, yet. Therefore, we have

$$\Pr[\mathbf{G}_{12}] = \Pr[\mathbf{G}_{11}].$$

Game $\mathbf{G}_{13}$: Replace the 3DH protocol. In this game we change the simulators behavior concerning the messages of the 3DH protocol. Instead of computing the messages of the 3DH protocol as in the real-world, the simulator uses its internal ideal AKE functionality $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ and its internal AKE simulator $\text{SIM}_{\mathsf{3DH}}$ to simulate the messages and outputs. In particular, this affects the messages U (cf. R.4), V (cf. Ra.3), the public and private keys y, Y (cf. PK.1 (c)) and X, x (cf. Ib.6), the computation of $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ (cf. Ra.3 and Rb.5) and queries to KDF_2 (K2.3).

A difficulty in this game arises when an honest client tries to retrieve a record that was initialized by a corrupt server. In that case, SIM cannot use $\text{SIM}_{\mathsf{3DH}}$ and $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ to output keys, as we cannot insert the DH share X chosen by the corrupt server into $\text{SIM}_{\mathsf{2HDH}}$. Instead SIM computes (u, U) as in $\mathbf{G}_{12}$ and upon receiving the message $(b_2, \ldots, \sigma_2)$ just copies the keys computed by $\text{SIM}_{\mathsf{3DH}}$ in Ra.9 if both parties agree an ikm and pre. Note that here it is crucial that SIM was able to extract the DH secret x chosen by the corrupt server as it otherwise cannot compute ikm for the client. If the parties do not agree on ikm and pre or SIM could not extract x, it chooses $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ uniformly at random such that they follow the same distribution as in $\mathbf{G}_{12}$.

Assume there is an environment $\mathcal{Z}^*$ that can distinguish between games $\mathbf{G}_{12}$ and $\mathbf{G}_{13}$. We can construct an environment $\mathcal{Z}_2$ that can distinguish between the real 3DH protocol and the simulated execution with $\text{SIM}_{\mathsf{3DH}}$ and $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. The environment $\mathcal{Z}_2$ internally runs the whole experiment including $\mathcal{F}$ and SIM with $\mathcal{Z}^*$, but whenever SIM would execute a operation of $\mathsf{3DH}$, it instead generates the corresponding output from the AKE experiment. If $\mathcal{Z}_2$ interacts with $\mathsf{3DH}$, it perfectly simulates $\mathbf{G}_{12}$ for $\mathcal{Z}^*$, and if it interacts with $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ and $\text{SIM}_{\mathsf{3DH}}$, it perfectly simulates $\mathbf{G}_{13}$ for $\mathcal{Z}^*$.

It follows that we have

$$|\Pr[\mathbf{G}_{13}] - \Pr[\mathbf{G}_{12}]| \leq \mathbf{Adv}^{\mathcal{F}_{\mathsf{AKE\text{-}KCI}}}_{\mathsf{3DH}, \text{SIM}_{\mathsf{3DH}}, \mathcal{Z}_2}(\lambda).$$

Game $\mathbf{G}_{14}$: Randomize $\mathsf{e\_cred}$. In this game we change the computation of the ciphertext $\mathsf{e\_cred}$. Whenever $\text{SIM}_{\mathsf{WBP}}$ simulates E for some honest cid, it chooses $\mathsf{e\_cred} \xleftarrow{\$} \{0,1\}^\lambda$ (see Ib.7). This ensures that $\mathsf{e\_cred}$ is independent of x and thus the adversary cannot learn anything about x from $\mathsf{e\_cred}$ as long as it does not guess the password of cid. If $\mathcal{A}$ guesses the password of cid, we program the random oracle KDF_1 such that it outputs $K_{\mathsf{mask}} := x \oplus \mathsf{e\_cred}$ (see K1.1 (a)). Since $\mathsf{e\_cred}$ was drawn uniformly at random, K_{mask} is a uniformly random value as well, which ensures that the distribution of K_{mask} does not change in this game.

From the information-theoretic security of the one-time-pad it follows that

$$\Pr[\mathbf{G}_{14}] = \Pr[\mathbf{G}_{13}].$$

Game $\mathbf{G}_{15}$: Skip verification of T_S. In this game we do not check the MAC T_S for validity anymore. Instead the simulator simply checks whether the messages (n_C, U, a_2) and $(b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, *)$ were honestly delivered by the adversary. If they were not honestly delivered, SIM lets $\mathcal{F}$ output FAIL to cid. $\mathbf{G}_{15}$ and $\mathbf{G}_{14}$ only deviate if the adversary modifies any part of these messages but the MAC T'_S it sends to cid is valid. We denote this as the event $\mathrm{E}_{\mathsf{MAC}}$. If this event occurs with non-negligible probability, we can construct an adversary against the $\mathsf{sEUF\text{-}CMA}$ security of MAC.

The reduction works as follows. Let q_{RET} be an upper bound on the number of retrievals. The reduction $\mathcal{B}_5$ guesses an index $i \in \{1, \ldots, q_{\mathrm{RET}}\}$ and runs the experiment internally. If the i-th retrieval is run by an honest cid, $\mathcal{B}_5$ does not compute T_S itself but instead lets T_S be computed by the oracle TAG, which implicitly sets $\mathsf{K}_S^{\mathsf{MAC}}$ to the key sampled by the challenger. This does not change the distribution of $\mathsf{K}_S^{\mathsf{MAC}}$ as it already is a uniformly random value as of $\mathbf{G}_{13}$. If the event $\mathrm{E}_{\mathsf{MAC}}$ occurs in the i-th retrieval, the reduction gives the value T_S that the client received as output to its challenger together with the corresponding message $\mathsf{H}_3(\mathsf{pre})$. Note that due to the changes introduced in $\mathbf{G}_8$, the environment $\mathcal{Z}^*$ also cannot somehow exploit collisions in the hash function H_3. One can see that $\mathcal{B}_5$ wins the $\mathsf{sEUF\text{-}CMA}$ experiment if it guessed i correctly and $\mathrm{E}_{\mathsf{MAC}}$ occurs in the i-th retrieval. Therefore, we have

$$|\Pr[\mathbf{G}_{15}] - \Pr[\mathbf{G}_{14}]| \leq q_{\mathrm{RET}} \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{B}_5}(\lambda).$$

Game $\mathbf{G}_{16}$: Skip verification of T_C. In this game we do not check the MAC T_C for validity anymore in retrievals by an honest cid. Instead the simulator simply checks whether the message T_C was honestly delivered by the adversary. If it was not honestly delivered, SIM produces the output FAIL to S. $\mathbf{G}_{16}$ and $\mathbf{G}_{15}$ only deviate if the adversary delivers some valid $T'_C \neq T_C$, which we denote as the event $\mathrm{E}'_{\mathsf{MAC}}$. If this event occurs with non-negligible probability, we can construct an adversary $\mathcal{B}_6$ against the $\mathsf{sEUF\text{-}CMA}$ security of MAC.

The reduction $\mathcal{B}_6$ essentially works just like $\mathcal{B}_5$ from $\mathbf{G}_{15}$. $\mathcal{B}_6$ guesses an index $i \in \{1, \ldots, q_{\mathrm{RET}}\}$ and lets T_C be computed by the oracle TAG in the i-th retrieval. Again, this does not change the distribution of $\mathsf{K}_C^{\mathsf{MAC}}$ as it already is a uniformly random value as of $\mathbf{G}_{13}$. If the event $\mathrm{E}'_{\mathsf{MAC}}$ occurs in the i-th retrieval, $\mathcal{B}_6$ gives the value T_C that the server received as output to its challenger together with the corresponding message $\mathsf{H}_3(\mathsf{pre} \parallel T_S)$. Thus, we have

$$|\Pr[\mathbf{G}_{16}] - \Pr[\mathbf{G}_{15}]| \leq q_{\mathrm{RET}} \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{B}_6}(\lambda).$$

Game $\mathbf{G}_{17}$: Skip verification of T_e. In this game, SIM does not check if T_e verifies when an honest cid receives a message $b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2$ and instead directly checks the passwords pw' that it still receives from $\mathcal{F}$ in this game. We claim that the distribution of the game does not change up to negligible difference with this modification.

Due to $\mathbf{G}_{15}$, the message $(b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, *)$ cannot be modified by the adversary and the values $\mathsf{e\_cred}, n_e$, and T_e must be equal to the ones that are stored by SIM. We now distinguish how the values stored by SIM could have been computed. First, they could have been computed by SIM in **??** when it simulated an initialization for cid, where the adversary honestly delivered E. Second, they could have been computed in an initialization by cid, where the corrupt server delivers some $\mathsf{E}' \neq \mathsf{E}$ to the HSM. Third, they could have been computed by the corrupt S in an initialization, where it impersonated cid.

In the first case, in $\mathbf{G}_{16}$, T_e would successfully verify if cid used the correct password. Thus, directly checking the password does not change the distribution. In the second and third case, in **??**, SIM tried to extract the password pw that was used to compute the values $\mathsf{e\_cred}', n_e'$, and T_e' obtained by decrypting E', in particular it must hold that T_e' verifies under the key K_{auth} derived from pw. If it was able to extract pw, we can again simply compare pw with the password pw' used by cid in this retrieval.

Now consider the case that SIM could not extract a password. Recall that in that case, it stored the error symbol $\perp$ as the password, which means that now in $\mathbf{G}_{17}$, cid will always output FAIL in any retrieval. Thus, we need argue that cid always outputs FAIL when it receives T_e' in $\mathbf{G}_{16}$ if SIM could not extract a password in **??**. If SIM was not able to extract pw, this must be due to T_e' not verifying under K_{auth}. If no key exists under which T_e' verifies, then in $\mathbf{G}_{16}$, cid will obviously also output FAIL in any retrieval where it receives T_e'. If T_e' verifies under some key $K_{\mathsf{auth}}{}' \neq K_{\mathsf{auth}}$, cid would not output FAIL if it ever derives $K_{\mathsf{auth}}{}'$ in some retrieval, meaning that $\mathbf{G}_{17}$ would deviate from $\mathbf{G}_{16}$. First, assume that $K_{\mathsf{auth}}{}'$ was not output by the random oracle KDF_1, yet. Then, the probability of cid ever deriving $K_{\mathsf{auth}}{}'$ in any retrieval can be bounded by $\frac{q_{\mathrm{RET}}}{2^\lambda}$, since the outputs of KDF_1 are drawn uniformly at random. If $K_{\mathsf{auth}}{}'$ was already output by KDF_1 in response to some query (ρ, n_e), then ρ was never output by the random oracle H_2 in response to some query (pw^*, h) before, as otherwise SIM would have extracted pw^* as the password used to compute T_e'. That means we can bound the probability of cid deriving ρ by $\frac{q_{\mathrm{RET}}}{2^\lambda}$, since the outputs of H_2 are drawn uniformly at random.

The argument for the third case, where the corrupt S impersonated cid in the initialization, follows analogously. Overall, we have

$$|\Pr[\mathbf{G}_{17}] - \Pr[\mathbf{G}_{16}]| \leq \frac{2q_{\mathrm{RET}}}{2^\lambda}.$$

Game $\mathbf{G}_{18}$: Output FAIL upon adversarial AE ciphertexts. In this game we let SIM immediately output FAIL to any cid if cid receives a ciphertext c that was not computed by SIM. $\mathbf{G}_{18}$ and $\mathbf{G}_{17}$ only deviate if a valid ciphertext is sent to cid that was not issued by the SIM before, which we denote as the event E_{AE}. If this happens with non-negligible probability, we can construct an adversary against the INT-CTXT security of AE.

Assume there is an environment $\mathcal{Z}^*$ that causes the event E_{AE} when interacting with $\mathcal{F}$ and SIM. Then, the reduction $\mathcal{B}_7$ internally runs the whole experiment including $\mathcal{F}$, SIM and $\mathcal{Z}^*$. Further, it guesses an index $i \xleftarrow{\$} \{1, \ldots, q_{\text{RET}}\}$ and in the i-th retrieval does not compute c by itself but instead asks the oracle ENC to compute c. Note that this implicitly programs the key shk to the key k chosen by the challenger but this does not change the distribution of shk as it is sampled uniformly at random by $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. Additionally, in $\mathbf{G}_{13}$ we have ensured that both parties use the same key shk, which means that if the event E_{AE} occurs in the i-th retrieval for some ciphertext c', c' is valid under the key k. Thus, if the event E_{AE} occurs, $\mathcal{B}_7$ wins the INT-CTXT experiment and we have

$$|\Pr[\mathbf{G}_{18}] - \Pr[\mathbf{G}_{17}]| \leq q_{\text{RET}} \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AE},\mathcal{B}_7}(\lambda).$$

Game $\mathbf{G}_{19}$: S encrypts 0 in c. In this game the the simulator computes c as $\mathsf{AE.Enc}(\mathsf{shk}, m)$, where m is a 0-string of appropriate length, in retrievals with honest clients. Assume there is an environment $\mathcal{Z}^*$ that can distinguish $\mathbf{G}_{19}$ and $\mathbf{G}_{18}$. We construct an adversary $\mathcal{B}_8$ against the IND-CPA security of AE as follows. We construct a sequence of games $\mathbf{G}^{(0)}_{18} := \mathbf{G}_{18}, \mathbf{G}^{(1)}_{18}, \ldots, \mathbf{G}^{(q_{\text{INIT}})}_{18} := \mathbf{G}_{19}$, where in $\mathbf{G}^{(i)}_{18}$ in the first i retrievals c is computed as an encryption of a 0-string using the oracle ENC and in the remaining retrievals it is computed as in $\mathbf{G}_{18}$. Because $\mathcal{Z}^*$ can distinguish between $\mathbf{G}_{19}$ and $\mathbf{G}_{18}$, there must be an index $i^* \in \{1, \ldots, q_{\text{RET}}\}$ such that $\mathcal{Z}^*$ can distinguish between $\mathbf{G}^{(i^*-1)}_{19}$ and $\mathbf{G}^{(i^*)}_{19}$. Now, in the i^*-th retrieval, $\mathcal{B}_8$ outputs $m_0 := \mathsf{e}$ and m_1 as a 0-string of the same length to its challenger to receive a challenge ciphertext c^* and sends c^* to cid. Note that due to $\mathbf{G}_{18}$ the environment cannot modify the ciphertext sent in any retrieval and hence $\mathcal{B}_8$ does not need a decryption oracle. Now, if the challenger encrypts m_0, the game is distributed exactly like $\mathbf{G}^{(i^*-1)}_{18}$ and if it encrypts m_1, it is distributed exactly like $\mathbf{G}^{(i^*)}_{18}$. Hence, we get

$$|\Pr[\mathbf{G}_{19}] - \Pr[\mathbf{G}_{18}]| \leq q_{\text{INIT}} \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AE},\mathcal{B}_8}(\lambda).$$

Game $\mathbf{G}_{20}$: Add INITC and RETC interfaces. In this game we introduce slightly modified interfaces INITC and RETC to $\mathcal{F}$ and add the interfaces INITS and RETS. The interfaces INITC and RETC act exactly as in $\mathcal{F}_{\mathsf{PPKR}}$ except that they still forward pw to SIM. These are only syntactical changes as the interfaces never give any output to parties. We have

$$\Pr[\mathbf{G}_{20}] = \Pr[\mathbf{G}_{19}]$$

Game $\mathbf{G}_{21}$: Let $\mathcal{F}$ produce output for clients. In this game SIM does not produce outputs directly to clients anymore and instead uses the COMPLETEINITC and COMPLETERETC interfaces of $\mathcal{F}$, which we add to $\mathcal{F}$ with the same code as in $\mathcal{F}_{\mathsf{PPKR}}$. Whenever SIM produced output to some honest cid in $\mathbf{G}_{20}$, it now issues the corresponding query to $\mathcal{F}$, in particular this affects .

In the initialization, clients always output K in $\mathbf{G}_{20}$, which SIM can simulate in $\mathbf{G}_{21}$ by sending $(\text{COMPLETEINITC}, \mathsf{ssid}, 1)$, making the changes in the initialization phase indistinguishable. The retrieval phase is more difficult to simulate as the client can output FAIL, DELREC, or K. To this end, we need to add a slightly modified COMPLETEINITS interface to $\mathcal{F}$ that only executes the Steps CIS.1 and CIS.3 to ensure that $\mathcal{F}$ stores FILE records that can be accessed in the retrieval. Similarly, we also introduce a slightly modified COMPLETERETS interface to ensure that $\mathcal{F}$ resets the ctr in the FILE records in a successful retrieval. The modified COMPLETERETS interface executes the Steps CRS.1, CRS.2, and CRS.3.

Let us now argue that the changes in the retrieval phase are indistinguishable as well. First, in this game $\mathcal{F}$ still forwards the passwords of clients to SIM, which makes it easy for SIM to determine whether a retrieval by an honest cid is successful and choose b for the COMPLETERETC query accordingly. For retrievals by a corrupt party, we distinguish whether the currently stored FILE record by SIM was created in an initialization by an honest or corrupt party. If it was an honest cid, indicated by the existence of a record $\langle \text{PROG}, \mathsf{kid}, [\rho], * \rangle$, the corrupt party must have guessed the password correctly in this retrieval. Otherwise it cannot decrypt $\mathsf{e\_cred}$, which means that it cannot learn anything about the clients DH secret x and cannot compute a valid T_C. Hence, a record $\langle \mathsf{H}_2, [\mathsf{pw}], *, \rho \rangle$ must exist and SIM can extract the password pw used by the corrupt party in this retrieval. If the currently stored FILE record was created in an initialization by a corrupt party, SIM does not need to extract a password since in that case it can always reset the counter in $\mathcal{F}$ with the COMPLETERETS or MALICIOUSRET interface.

Finally, we need to ensure that ctr is decremented in any retrieval. If S is honest it is decremented in the INITS interface. If S is corrupt, SIM issues RETS on behalf of the corrupt server in Ra.8.

In summary, the simulator ensures that clients receive the same outputs as in $\mathbf{G}_{20}$ by (1) using the password it receives from $\mathcal{F}$ to determine whether retrievals should be successful and (2) ensuring that the ctr values in $\mathcal{F}$ are always updated correctly by decrementing it in every initialization and resetting it to 10 whenever there is a successful retrieval. Therefore, the distribution of $\mathbf{G}_{21}$ is the same as in $\mathbf{G}_{20}$ and we have

$$\Pr[\mathbf{G}_{21}] = \Pr[\mathbf{G}_{20}].$$

Game $\mathbf{G}_{22}$: Let $\mathcal{F}$ produce output for the server. In this game, we change the simulator such that it uses the CompleteInitS and CompleteRetS interfaces to produce output to the server. To this end, we add the missing steps from $\mathcal{F}_{\mathsf{PPKR}}$ of these interfaces to $\mathcal{F}$. Since in $\mathbf{G}_{21}$ the simulator already used these interfaces in successful initializations and retrievals, we only need to add the queries in failed initializations and retrievals. Here, Sim always sets $b = 0$, which ensures that S receives the output Fail, and we have

$$\Pr[\mathbf{G}_{22}] = \Pr[\mathbf{G}_{21}].$$

Game $\mathbf{G}_{23}$: Remove password forwarding from $\mathcal{F}$. In this game, we finally remove the password forwarding of $\mathcal{F}$. In $\mathbf{G}_{22}$, Sim used the password inputs to determine whether recoveries should be successful. But instead of comparing the passwords by itself, it can also use the output *match* from $\mathcal{F}$, which gives the exact same information. Therefore, the distribution of the game does not change:

$$\Pr[\mathbf{G}_{23}] = \Pr[\mathbf{G}_{22}].$$

Furthermore, we have now arrived at the point, where $\mathcal{F}$ executes exactly the code of $\mathcal{F}_{\mathsf{PPKR}}$ and Sim executes the code of $\text{Sim}_{\mathsf{WBP}}$. We have thus arrived at the ideal world execution, and summing up the terms from the game hops gives the bound given in Theorem 5.

□

Simulator $\textsc{Sim}_{\mathsf{WBP}}$, part 1

$\textsc{Sim}_{\mathsf{WBP}}$ stores a map $\mathsf{aids}[\cdot]$ that stores the aid belonging to cid. *Wait for* X means that $\textsc{Sim}_{\mathsf{WBP}}$ does not proceed to the next instruction before receiving X and meanwhile gives back activation to $\mathcal{A}$. Once it receives X, it first proceeds with the instructions on input X and then continues at the instruction, where it waited for X. If a record cannot be retrieved, the query is ignored. For brevity we omit session identifier sid from all inputs, outputs, and records.

On GetPK from anyone to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$:

PK.1 If no record $\langle \mathrm{PK}, [\mathsf{pk}], * \rangle$ exists:
- (a) $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{Sig.KGen}(1^\lambda)$ $(\mathbf{G}_1)$
- (b) $(\mathsf{sk}_{\mathsf{Enc}}, \mathsf{pk}_{\mathsf{Enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ $(\mathbf{G}_1)$
- (c) Give input $(\textsc{Init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 1 of $\textsc{Sim}_{\mathsf{3DH}}$, and then set Y to the values A generated in that step $(\mathbf{G}_{13})$
- (d) Set $\mathsf{sk} = \{\mathsf{sk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Enc}}, \bot, \mathsf{K}_{\mathsf{Enc}}\}$ and $\mathsf{pk} = \{\mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Enc}}, Y\}$ and store $\langle \mathrm{PK}, \mathsf{pk}, \mathsf{sk} \rangle$ $(\mathbf{G}_1)$

PK.2 Reply with pk $(\mathbf{G}_1)$

On corrupt from $\mathcal{A}$ to S:

C.1 For each record $\langle \textsc{File}, [\mathsf{cid}], *, *, *, *, *, *, *, * \rangle$, choose a fresh $\mathsf{aid} \xleftarrow{\$} \{0,1\}^\lambda$, set $\mathsf{aids}[\mathsf{cid}] \coloneqq \mathsf{aid}$, and overwrite cid with aid in the File record $(\mathbf{G}_2)$

C.2 Output $\mathsf{aids}[\cdot]$ $(\mathbf{G}_1)$

On $(\textsc{InitC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest cid

I.1 Choose a fresh $\mathsf{kid} \xleftarrow{\$} \{0,1\}^\lambda$ that has not been used in any other initialization $(\mathbf{G}_{10})$

I.2 Give input $(\textsc{Eval}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}, 0)$ to $\mathcal{F}_{\mathsf{OPRF}}$ from cid. This triggers Step 5 of $\textsc{Sim}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{cid}, \mathsf{ssid}, a)$ $(\mathbf{G}_{10})$

I.3 Record $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, a, \mathsf{kid}, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot \rangle$ and send (ssid, a) as message from cid to S $(\mathbf{G}_1)$

On $(\textsc{InitS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest S

I.4 If S honest, wait for (ssid, a_1) from cid to S $(\mathbf{G}_1)$ // See (ssid, a_1) interface

On (ssid, a_1) from $\mathcal{A}$ to S on behalf of cid: // any cid, honest S

Ia.1 If S is honest, wait for $(\textsc{InitS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_1)$

Ia.2 If cid is corrupt:
- (a) Choose a fresh $\mathsf{kid} \xleftarrow{\$} \{0,1\}^\lambda$ that has not been used in any other initialization $(\mathbf{G}_{10})$
- (b) Record $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, a, \mathsf{kid}, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot \rangle$ $(\mathbf{G}_1)$

Ia.3 If cid is honest, retrieve $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, *, [\mathsf{kid}], *, *, *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$

Ia.4 Delete any existing record $\langle \textsc{File}, \mathsf{cid}, *, *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$

Ia.5 Give input $(\textsc{Init}, \mathsf{sid}, \mathsf{kid})$ to $\mathcal{F}_{\mathsf{OPRF}}$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 2 of $\textsc{Sim}_{\mathsf{2HDH}}$ $(\mathbf{G}_{10})$ // Create new OPRF key

Ia.6 Give input $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{OPRF}}$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 6 of $\textsc{Sim}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{ssid}, b)$ $(\mathbf{G}_{10})$

Ia.7 Compute $n_1 \xleftarrow{\$} \{0,1\}^\lambda$ and $\sigma_1 \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{Sig}}, b_1 \parallel n_1)$. Overwrite the entries 7-10 in $\langle \textsc{Init}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, *, * \rangle$ with a_1, b_1, n_1, σ_1. Send $(\mathsf{ssid}, b_1, n_1, \sigma_1)$ from S to cid $(\mathbf{G}_1)$. Abort if another record $\langle \textsc{Init}, *, *, *, *, *, *, *, n_1, *, *, *, *, * \rangle$ exists $(\mathbf{G}_5)$

Figure 4.18: Simulator $\textsc{Sim}_{\mathsf{WBP}}$ for WBP, part 1.

Simulator $\text{SIM}_{\mathsf{WBP}}$, part 2

On $(\text{INITS}, \mathsf{ssid}, \mathsf{aid}, \mathsf{aid}', a_1^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ on behalf of S:

Ia.8 Run HSM code up to checking if aid is fresh. If it is not fresh, send $(\text{INITRES}, \mathsf{aid}, \text{FAIL})$ to S, and else continue $(\mathbf{G}_1)$

Ia.9 If no record $\langle \text{INIT}, \mathsf{ssid}, [\mathsf{cid}], *, [\mathsf{kid}], *, *, *, *, *, *, *, * \rangle$ exists:

(a) Choose a fresh $\mathsf{kid} \xleftarrow{\$} \{0,1\}^\lambda$ that has not been used in any other initialization $(\mathbf{G}_{10})$

(b) Record $\langle \text{INIT}, \mathsf{ssid}, \bot, *, \mathsf{kid}, \bot, a_1^*, \bot, \bot, \bot, \bot, \bot, \bot, \bot \rangle$ and set $\mathsf{cid} := \bot$ $(\mathbf{G}_1)$

Ia.10 Overwrite the sixth entry of the record $\langle \text{INIT}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, * \rangle$ with aid $(\mathbf{G}_1)$

Ia.11 Delete any existing record $\langle \text{FILE}, \mathsf{aid}', *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$

Ia.12 If $\mathsf{cid} \neq \bot$, set $\mathsf{aids}[\mathsf{cid}] := \mathsf{aid}$ $(\mathbf{G}_2)$

Ia.13 Execute Steps Ia.5 - Ia.7 where the output is returned to S instead of cid $(\mathbf{G}_{10})$

On $(\mathsf{ssid}, b_1, n_1, \sigma_1)$ from $\mathcal{A}$ to C on behalf of S: // honest cid

Ib.1 Retrieve $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, *, [\mathsf{kid}], [\mathsf{aid}], *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$

Ib.2 If S is not honest, give input $(\text{INITS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{aid})$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ or if $\mathcal{F}_{\mathsf{PPKR}}$ ignores the input, continue below $(\mathbf{G}_{21})$ // Ensures that we can produce output for cid

Ib.3 [Forged signature] If no record $\langle \text{INIT}, [\mathsf{ssid}'], *, *, *, *, *, b_1, n_1, \sigma_1, *, *, * \rangle$ exists $(\mathbf{G}_6)$, send $(\text{COMPLETEINITC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{21})$

Ib.4 If $\mathsf{ssid}' = \mathsf{ssid}$, give input $(\text{RCVCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathsf{cid}, \mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}})$ to $\mathcal{F}_{\mathsf{OPRF}}$, which outputs $(\text{EVALOUT}, \mathsf{sid}, \mathsf{ssid}, \rho)$ $(\mathbf{G}_{10})$

Ib.5 If $\mathsf{ssid}' \neq \mathsf{ssid}$, sample $\rho \xleftarrow{\$} \{0,1\}^\lambda$ $(\mathbf{G}_{10})$

Ib.6 Send $(\mathsf{sid}, \text{INIT})$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ from cid, which triggers Step 1 of $\text{SIM}_{\mathsf{3DH}}$, and then set (x, X) to the values (a, A) generated in that step $(\mathbf{G}_{13})$

Ib.7 Compute $\mathsf{e\_cred} \xleftarrow{\$} \{0,1\}^\lambda$ $(\mathbf{G}_{14})$, $K_{\mathsf{auth}} \xleftarrow{\$} \{0,1\}^\lambda$ $(\mathbf{G}_{14})$, $n_e \xleftarrow{\$} \{0,1\}^\lambda$, and $T_e \xleftarrow{\$} \mathsf{MAC.Tag}(K_{\mathsf{auth}}, Y \,\|\, n_e, \mathsf{e\_cred})$ $(\mathbf{G}_1)$, and $\mathsf{E} \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{Enc}}, m)$, where m is a 0-string of appropriate length and record $\langle \mathsf{E}, X \,\|\, \mathsf{e\_cred} \,\|\, n_e \,\|\, T_e \rangle$ $(\mathbf{G}_7)$ and $\langle \text{PROG}, \mathsf{kid}, \rho, K_{\mathsf{auth}} \rangle$ $(\mathbf{G}_{14})$

Ib.8 Abort if another record $\langle \text{FILE}, *, *, *, *, *, *, *, n_e, *, * \rangle$ exists $(\mathbf{G}_5)$

Ib.9 Overwrite the entries 11-14 in the record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, *, \mathsf{kid}, *, *, *, *, *, *, *, * \rangle$ with b_1, n_1, x, E and send $(\mathsf{ssid}, \mathsf{E})$ as message from cid to S $(\mathbf{G}_1)$ and $(\text{COMPLETEINITC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{21})$

On $(\mathsf{ssid}, \mathsf{E})$ from $\mathcal{A}$ to S on behalf of cid:

IE.1 Retrieve $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, [a_1], [\mathsf{kid}], *, [a_1^*], [b_1], [n_1], *, [b_1^*], [n_1^*], *, * \rangle$ $(\mathbf{G}_1)$

IE.2 If cid is corrupt, run the HSM code up to determining out $(\mathbf{G}_1)$

(a) If $\mathsf{out} = \text{FAIL}$, send $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{22})$

(b) If $\mathsf{out} = \text{SUCC}$, Let $\mathsf{e} \,\|\, X \,\|\, \mathsf{tr} \,\|\, \mathsf{e\_cred} \,\|\, n_e \,\|\, T_e$ denote the decryption of E. Search for a record $\langle \mathsf{H}_2, [\mathsf{pw}], *, [\rho] \rangle$ marked CONSISTENT such that $\mathsf{MAC.Vrfy}(K_{\mathsf{auth}}, Y \,\|\, n_e \,\|\, \mathsf{e\_cred}, T_e) = 1$ holds for $(K_{\mathsf{export}}, K_{\mathsf{auth}}, K_{\mathsf{mask}}) := \mathsf{KDF}_1(\rho, n_e)$ $(\mathbf{G}_{12})$. If more than one such record exists, abort the simulation $(\mathbf{G}_{11})$. If exactly one such record exists, give input $(\text{MALICIOUSINIT}, \mathsf{aid}, \mathsf{pw}, K)$ to $\mathcal{F}_{\mathsf{PPKR}}$ for $K := \mathsf{AE.Dec}(K_{\mathsf{export}}, \mathsf{e})$. If no such record exists, give input $(\text{MALICIOUSINIT}, \mathsf{aid}, \bot, \bot)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{12})$. Record $\langle \text{FILE}, \mathsf{cid}, \mathsf{ssid}, \mathsf{kid}, \mathsf{e}, X, \mathsf{e\_cred}, n_e, T_e, 10 \rangle$ overwriting any existing $\langle \text{FILE}, \mathsf{cid}, *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$ and send $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{22})$

IE.3 If cid is honest:

(a) If $a_1 \neq a_1^*$, $b_1 \neq b_1^*$, or $n_1 \neq n_1^*$ $(\mathbf{G}_9)$, set $b := 1$, else set $b := 0$ $(\mathbf{G}_{22})$

(b) Retrieve $\langle \mathsf{E}, [X] \,\|\, [\mathsf{e\_cred}] \,\|\, [n_e] \,\|\, [T_e] \rangle$ $(\mathbf{G}_7)$, record $\langle \text{FILE}, \mathsf{cid}, \mathsf{ssid}, \mathsf{kid}, \mathsf{e}, X, \mathsf{e\_cred}, n_e, T_e, 10 \rangle$ overwriting any existing $\langle \text{FILE}, \mathsf{cid}, *, *, *, *, *, *, *, * \rangle$ $(\mathbf{G}_1)$, and send $(\text{COMPLETEINITS}, \mathsf{ssid}, b)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{22})$

Figure 4.19: Simulator $\text{SIM}_{\mathsf{WBP}}$ for WBP, part 2.

Simulator $\text{SIM}_{\mathsf{WBP}}$, part 3

On $(\mathsf{ssid}, \mathsf{E})$ from S to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ on behalf of S:

IE.4 Retrieve $\langle \text{INIT}, \mathsf{ssid}, [\mathsf{cid}], [a_1], [\mathsf{kid}], [\mathsf{aid}], [a_1^*], [b_1], [n_1], *, [b_1^*], [n_1^*], *, [\mathsf{E}'] \rangle$ $(\mathbf{G}_1)$

IE.5 [Honest delivery of E] If $\mathsf{E} = \mathsf{E}'$, execute IE.3 $(\mathbf{G}_7)$ and set $\mathsf{aids}[\mathsf{cid}] \coloneqq \mathsf{aid}$ $(\mathbf{G}_2)$

IE.6 If $\mathsf{E} \neq \mathsf{E}'$ $(\mathbf{G}_7)$, run HSM code up to determining out $(\mathbf{G}_1)$

(a) If $\mathsf{out} = \text{FAIL}$, send $(\text{INITRES}, \mathsf{ssid}, \text{FAIL})$ to S as output of the HSM, else continue $(\mathbf{G}_1)$

(b) Let $\mathsf{e} \parallel X \parallel \mathsf{tr} \parallel \mathsf{e\_cred} \parallel n_e \parallel T_e$ denote the decryption of E. Search for a record $\langle \mathsf{H}_2, [\mathsf{pw}], *, [\rho] \rangle$ marked CONSISTENT such that $\mathsf{MAC.Vrfy}(K_{\mathsf{auth}}, Y \parallel n_e \parallel \mathsf{e\_cred}, T_e) = 1$ holds for $(K_{\mathsf{export}}, K_{\mathsf{auth}}, K_{\mathsf{mask}}) \coloneqq \mathsf{KDF}_1(\rho, n_e)$ $(\mathbf{G}_{12})$. If more than one such record exists, abort the simulation $(\mathbf{G}_{11})$. If exactly one such record exists, give input $(\text{MALICIOUSINIT}, \mathsf{aid}, \mathsf{pw}, K)$ to $\mathcal{F}_{\mathsf{PPKR}}$ for $K \coloneqq \mathsf{AE.Dec}(K_{\mathsf{export}}, \mathsf{e})$. If no such record exists, give input $(\text{MALICIOUSINIT}, \mathsf{aid}, \perp, \perp)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{12})$

(c) Overwrite the 13th entry of the record $\langle \text{INIT}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, *, *, * \rangle$ with $\mathsf{e\_cred} \oplus K_{\mathsf{mask}}$ $(\mathbf{G}_{12})$

(d) If $\mathsf{cid} \neq \perp$, set $\mathsf{aids}[\mathsf{cid}] \coloneqq \mathsf{aid}$ $(\mathbf{G}_2)$

(e) Record $\langle \text{FILE}, \mathsf{aid}, \mathsf{ssid}, \mathsf{kid}, \mathsf{e}, X, \mathsf{e\_cred}, n_e, T_e, 10 \rangle$ $(\mathbf{G}_1)$

On $(\text{RETC}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest cid

R.1 If S is honest, retrieve $\langle \text{FILE}, \mathsf{cid}, [\mathsf{ssid}'], [\mathsf{kid}], *, *, *, *, *, * \rangle$, and otherwise retrieve $\langle \text{FILE}, \mathsf{aids}[\mathsf{cid}], [\mathsf{ssid}'], [\mathsf{kid}], *, *, *, *, *, * \rangle$ $(\mathbf{G}_2)$. If no such record exists, choose a fresh $\mathsf{kid} \xleftarrow{\$} \{0,1\}^\lambda$ that has not been used in any initialization $(\mathbf{G}_{10})$

R.2 Give input $(\text{EVAL}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid}, \mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}, 0)$ to $\mathcal{F}_{\mathsf{OPRF}}$ from cid. This triggers Step 5 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{cid}, \mathsf{ssid}, a)$ $(\mathbf{G}_{10})$

R.3 If no record $\langle \text{INIT}, \mathsf{ssid}', \mathsf{cid}, *, *, *, *, *, *, *, *, *, *, *, * \rangle$ exists, compute $u \xleftarrow{\$} \mathbb{Z}_q$ and $U \coloneqq g^u$ and record $\langle \text{DH}, \mathsf{ssid}, \perp, \perp \rangle$ $(\mathbf{G}_{13})$ // Corrupt S maliciously initialized for cid and we cannot use $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ as it does not know the DH public key

R.4 Otherwise, give input $(\text{NEWSESSION}, \mathsf{sid}, \mathsf{ssid}, \mathsf{cid}, \mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}})$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ from cid. This triggers Step 2 of $\text{SIM}_{\mathsf{3DH}}$, which outputs $(U, X, \perp)$ $(\mathbf{G}_{13})$

R.5 Compute $n_C \xleftarrow{\$} \{0,1\}^\lambda$ and record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, (n_C, U, a), \mathit{match}, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp \rangle$ and send $(\mathsf{ssid}, n_C, U, a)$ as message from cid to S $(\mathbf{G}_1)$

R.6 Abort if another record $\langle \text{RET}, *, *, (n_C, *, *), *, *, *, *, *, *, *, *, * \rangle$ exists $(\mathbf{G}_5)$

On $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest S

R.7 If a record $\langle \text{RET}, \mathsf{ssid}, *, *, \perp, *, *, *, *, *, *, *, * \rangle$ exists, overwrite $\perp$ with match $(\mathbf{G}_1)$

R.8 Otherwise, record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, \perp, \mathit{match}, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp \rangle$ $(\mathbf{G}_1)$

R.9 If S honest, wait for $(\mathsf{ssid}, n_C, U, a_2)$ from cid to S $(\mathbf{G}_1)$ // See $(\mathsf{ssid}, n_C, U, a_2)$ interface

Figure 4.20: Simulator $\text{SIM}_{\mathsf{WBP}}$ for WBP, part 3.

Simulator $\text{SIM}_{\mathsf{WBP}}$, part 4

On $(\mathsf{ssid}, n_C, U, a_2)$ from $\mathcal{A}$ to S on behalf of cid: // honest S

Ra.1 If S honest, wait for $(\text{RETS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_1)$

Ra.2 Retrieve $\langle \text{FILE}, \mathsf{cid}, *, [\mathsf{kid}], *, [X], [\mathsf{e\_cred}], [n_e], [T_e], * \rangle$. If no such record exists, send $(\text{COMPLETERETS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_1)$

Ra.3 Run HSM code on input $(\mathsf{ssid}, \mathsf{aid}, n_C, U, a_2)$ $(\mathbf{G}_1)$ with $\mathsf{aid} := \mathsf{cid}$ $(\mathbf{G}_2)$ and the following modifications:

- To compute b_2 give input $(\text{SNDRCOMPLETE}, \mathsf{sid}, \mathsf{kid}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{OPRF}}$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 6 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{ssid}, b_2)$ $(\mathbf{G}_{10})$
- To compute $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ give input $(\text{NEWSESSION}, \mathsf{sid}, \mathsf{ssid}, \mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. This triggers Step 2 of $\text{SIM}_{\mathsf{3DH}}$, which outputs $(V, Y, \perp)$. Then send (U, X, pre) to $\text{SIM}_{\mathsf{3DH}}$ as a message from cid to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 3 of $\text{SIM}_{\mathsf{3DH}}$, which lets $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{pre}, s)$. Interpret s as $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ $(\mathbf{G}_{13})$

If the output of the HSM code is $(\mathsf{ssid}, \text{DELREC})$, send $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{22})$ and $(\mathsf{ssid}, \text{DELREC})$ from S to cid. Otherwise continue $(\mathbf{G}_1)$

Ra.4 Set $m := (b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2)$. Overwrite the entries 6 and 8-10 of the record $\langle \text{RET}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, * \rangle$ with kid, (n_C, U, a_2), $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$, and m and send (ssid, m) from S to cid $(\mathbf{G}_1)$

Ra.5 Abort if another record $\langle \text{RET}, *, *, *, *, *, *, *, *, m', *, *, * \rangle$ exists with $m' = (*, *, *, *, n_S, *, *, *)$ $(\mathbf{G}_5)$

On $(\mathsf{ssid}, \mathsf{aid}, n_C, U, a_2)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ on behalf of S:

Ra.6 If no record $\langle \text{FILE}, \mathsf{aid}, *, [\mathsf{kid}], *, [X], [\mathsf{e\_cred}], [n_e], [T_e], * \rangle$ exists, send $(\text{RETRES}, \mathsf{ssid}, \text{FAIL})$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ to S $(\mathbf{G}_1)$

Ra.7 If no record $\langle \text{RET}, \mathsf{ssid}, [\mathsf{cid}], *, *, *, *, *, *, *, *, *, *, * \rangle$ with $\mathsf{cid} \neq \perp$ exists, record $\langle \text{RET}, \mathsf{ssid}, \perp, \perp, \perp, \perp, \mathsf{aid}, \perp, \perp, \perp, \perp, \perp, \perp \rangle$ and set $\mathsf{cid} = \perp$ $(\mathbf{G}_1)$

Ra.8 Give input $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}', \mathsf{aid}, \perp, \perp, \perp)$ to $\mathcal{F}_{\mathsf{PPKR}}$, where $\mathsf{cid}' := \mathsf{cid}$ if $\mathsf{cid} \neq \perp$ and $\mathsf{cid}' := \mathsf{aid}$ otherwise. On response $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}', \mathit{match})$ overwrite the fifth entry of the record $\langle \text{RET}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, * \rangle$ with match $(\mathbf{G}_{23})$

Ra.9 Execute Steps Ra.3 - Ra.5, where $\mathsf{cid} := \mathsf{aid}$ if $\mathsf{cid} = \perp$, and returning the output to S instead of cid $(\mathbf{G}_{10})$

Ra.10 If a record $\langle \text{DH}, \mathsf{ssid}, *, * \rangle$ exists: $(\mathbf{G}_{13})$

(a) Let v denote the value chosen in Step 2 of $\text{SIM}_{\mathsf{3DH}}$ during Ra.3 with $V = g^v$ $(\mathbf{G}_{13})$

(b) Overwrite the last two entries of $\langle \text{DH}, \mathsf{ssid}, *, * \rangle$ with $((U^v, U^y, X^v), (\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk}))$ $(\mathbf{G}_{13})$

On $(\mathsf{ssid}, \text{DELREC})$ from $\mathcal{A}$ to cid on behalf of S:

D.1 If no record $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, *, *, *, *, *, *, *, \perp, \perp, \perp \rangle$ exists, ignore this query $(\mathbf{G}_1)$

D.2 If S is corrupt, give input $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{cid}, \perp, \perp, \perp)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$ continue below $(\mathbf{G}_{21})$

D.3 Send $(\text{COMPLETERETC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{21})$

Figure 4.21: Simulator $\text{SIM}_{\mathsf{WBP}}$ for WBP, part 4.

Simulator $\textsc{Sim}_{\mathsf{WBP}}$, part 5

On $(\mathsf{ssid}, b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2)$ from $\mathcal{A}$ to cid on behalf of S:

Rb.1 Retrieve $\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, [m_1], [match], [\mathsf{kid}], [\mathsf{aid}], [m_1^*], *, [m_2], *, *, * \rangle$. If no such record exists, ignore the query ($\mathbf{G}_1$)

Rb.2 If S is corrupt, give input $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{aid}, \bot, \bot, \bot)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetS}, \mathsf{ssid}, \mathsf{aid}, match')$ from $\mathcal{F}_{\mathsf{PPKR}}$ set $match \coloneqq match'$ and overwrite $match$ with $match'$ in the record retrieved above ($\mathbf{G}_{23}$)

Rb.3 [mismatching password or preamble] If $match = 0$ ($\mathbf{G}_{23}$), $m_1 \neq m_1^*$, or $(b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2) \neq m_2$ ($\mathbf{G}_6$, $\mathbf{G}_{15}$), send $(\textsc{CompleteRetC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ ($\mathbf{G}_{21}$), else continue ($\mathbf{G}_6$)

Rb.4 If a record $\langle \textsc{Dh}, \mathsf{ssid}, [\mathsf{ikm}], ([\mathsf{K}_C^{\mathsf{MAC}}], *, *) \rangle$ exists: ($\mathbf{G}_{13}$)

(a) Retrieve $\langle \textsc{Init}, *, *, \mathsf{kid}, *, *, *, *, *, *, *, [x], * \rangle$ ($\mathbf{G}_{13}$)

(b) If $\mathsf{ikm} \neq (V^u, Y^u, V^x)$ ($\mathbf{G}_{13}$), send $(\textsc{CompleteRetC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ ($\mathbf{G}_{21}$), else continue ($\mathbf{G}_{13}$)

Rb.5 If no record $\langle \textsc{Dh}, \mathsf{ssid}, *, * \rangle$ exists, send (V, Y, pre) to $\textsc{Sim}_{\mathsf{3DH}}$ as a message from cid to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$. This triggers Step 3 of $\textsc{Sim}_{\mathsf{3DH}}$, which lets $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{pre}, s)$. Interpret s as $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ ($\mathbf{G}_{13}$)

Rb.6 Compute $T_C \xleftarrow{\$} \mathsf{MAC.Tag}(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{H}_3(\mathsf{pre}, T_S))$. Overwrite the tenth entry of record retrieved in Rb.1 with T_C and send (ssid, T_C) from cid to S ($\mathbf{G}_1$)

On (ssid, T_C) from $\mathcal{A}$ to S on behalf of cid: // honest S

RT.1 If cid is honest: ($\mathbf{G}_{16}$)

(a) If no record $\langle \textsc{Ret}, \mathsf{ssid}, *, *, *, *, *, *, (*, *, [\mathsf{shk}]), *, T_C, *, * \rangle$ exists ($\mathbf{G}_{16}$), send $(\textsc{CompleteRetS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ ($\mathbf{G}_{22}$), else continue ($\mathbf{G}_{16}$)

(b) Compute $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, m)$, where m is a 0-string of appropriate length ($\mathbf{G}_{19}$)

RT.2 If cid is corrupt:

(a) Retrieve $\langle \textsc{Ret}, \mathsf{ssid}, *, *, *, [\mathsf{kid}], *, *, ([\mathsf{K}_C^{\mathsf{MAC}}], [\mathsf{K}_S^{\mathsf{MAC}}], [\mathsf{shk}]), [m], *, *, * \rangle$ with $m = (*, *, *, *, *, *, [T_S], *)$ ($\mathbf{G}_1$)

(b) If $\mathsf{MAC.Vrfy}(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{H}_3(\mathsf{pre} \| T_S), T_C) = 0$, give input $(\textsc{RetC}, \mathsf{ssid}, \bot)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ send $(\textsc{CompleteRetS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. ($\mathbf{G}_{22}$). Otherwise continue below. ($\mathbf{G}_1$)

(c) If records $\langle \textsc{Prog}, \mathsf{kid}, [\rho], * \rangle$ and $\langle \mathsf{H}_2, [\mathsf{pw}], *, \rho \rangle$ exist, give input $(\textsc{RetC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetC}, \mathsf{ssid}, \mathsf{cid}, match)$, send $(\textsc{CompleteRetC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetRes}, \mathsf{ssid}, K)$, compute $\mathsf{e} \xleftarrow{\$} \mathsf{AE.Enc}(K_{\mathsf{export}}, K)$ and $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, \mathsf{e})$ for $(K_{\mathsf{export}}, *, *) \coloneqq \mathsf{KDF}_1(\rho, n_e)$ ($\mathbf{G}_{??}$)

(d) Otherwise, give input $(\textsc{RetC}, \mathsf{ssid}, \bot)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetC}, \mathsf{ssid}, \mathsf{cid}, match)$, compute $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, \mathsf{e})$ ($\mathbf{G}_{19}$)

RT.3 Overwrite the last entry of the record $\langle \textsc{File}, \mathsf{cid}, *, *, *, *, *, *, *, *, * \rangle$ with 10 and the last two entries of the record $\langle \textsc{Ret}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, * \rangle$ with (T_C, c). Send (ssid, c) from S to cid ($\mathbf{G}_1$) and $(\textsc{CompleteRetS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ ($\mathbf{G}_{21}$)

Figure 4.22: Simulator $\textsc{Sim}_{\mathsf{WBP}}$ for WBP, part 5.

Simulator $\text{SIM}_{\mathsf{WBP}}$, part 6

On (ssid, T_C) from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ on behalf of S: // corrupt S

RT.4 Retrieve $\langle \text{RET}, \mathsf{ssid}, *, *, [match], [\mathsf{kid}], [\mathsf{aid}], *, ([\mathsf{K}_C^{\mathsf{MAC}}], *, [\mathsf{shk}], *, [T'_C], *, *)\rangle$ $(\mathbf{G}_1)$

RT.5 If $T_C = T'_C$ $(\mathbf{G}_{16})$ and compute $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, m)$, where m is a 0-string of appropriate length $(\mathbf{G}_{19})$

RT.6 If $T_C \neq T'_C$: $(\mathbf{G}_{16})$

(a) If $\mathsf{MAC.Vrfy}(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{H}_3(\mathsf{pre} \parallel T_S), T_C) = 0$, send $(\text{RETRES}, \mathsf{ssid}, \text{FAIL})$ from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ to S, else continue below $(\mathbf{G}_1)$

(b) If a record $\langle \text{PROG}, \mathsf{kid}, [\rho], *\rangle$ and $\langle \mathsf{H}_2, [\mathsf{pw}], *, \rho\rangle$ exist, send $(\text{MALICIOUSRET}, \mathsf{cid}, \mathsf{pw}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$, where cid is defined such that $\mathsf{aids}[\mathsf{cid}] = \mathsf{aid}$ holds. On response K from $\mathcal{F}_{\mathsf{PPKR}}$ compute $\mathsf{e} \xleftarrow{\$} \mathsf{AE.Enc}(K_{\mathsf{export}}, K)$ and $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, \mathsf{e})$ for $(K_{\mathsf{export}}, *, *) := \mathsf{KDF}_1(\rho, n_e)$ $(\mathbf{G}_{??})$

(c) Otherwise, send $(\text{MALICIOUSRET}, \mathsf{aid}, \perp, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$, retrieve $\langle \text{FILE}, \mathsf{aid}, *, *, [\mathsf{e}], *, *, *, *, *\rangle$ and compute $c \xleftarrow{\$} \mathsf{AE.Enc}(\mathsf{shk}, \mathsf{e})$ $(\mathbf{G}_1)$

RT.7 Overwrite the last entry of the record $\langle \text{FILE}, \mathsf{aid}, *, *, *, *, *, *, *, *\rangle$ with 10 and the last two entries of the record $\langle \text{RET}, \mathsf{ssid}, *, *, *, *, *, *, *, *, *, *, *\rangle$ with (T_C, c) and send (ssid, c) from $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{WBP}}$ to $\mathcal{A}$ $(\mathbf{G}_1)$ and $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{21})$

On (ssid, c) from $\mathcal{A}$ to cid on behalf of S: // honest cid

Rc.1 Retrieve $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, *, *, *, *, *, *, *, *, *, [c']\rangle$ $(\mathbf{G}_1)$

Rc.2 Send $(\text{COMPLETERETC}, \mathsf{ssid}, c \stackrel{?}{=} c')$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{18}, \mathbf{G}_{21})$

On random oracle query $\mathsf{H}_1(\mathsf{pw})$ from $\mathcal{A}$:

H1.1 Send $\mathsf{H}_1(\mathsf{pw})$ from $\mathcal{A}$ to $\text{SIM}_{\mathsf{2HDH}}$. This triggers Step 4 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs h. Output h to $\mathcal{A}$ $(\mathbf{G}_{10})$

On random oracle query $\mathsf{H}_2(\mathsf{pw}, h)$ from $\mathcal{A}$:

H2.1 If a record $\langle \mathsf{H}_2, \mathsf{pw}, h, [\rho]\rangle$ exists, output ρ $(\mathbf{G}_1)$

H2.2 Send $\mathsf{H}_2(\mathsf{pw}, h)$ to $\text{SIM}_{\mathsf{2HDH}}$. This triggers Step 8 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs ρ $(\mathbf{G}_{10})$

H2.3 If a record $\langle \mathsf{H}_2, *, *, \rho\rangle$ exists, abort $(\mathbf{G}_3)$

H2.4 Record $\langle \mathsf{H}_2, \mathsf{pw}, h, \rho\rangle$ and mark it CONSISTENT if ρ was produced by $\mathcal{F}_{\mathsf{OPRF}}$ during Step 8 of $\text{SIM}_{\mathsf{2HDH}}$ $(\mathbf{G}_{10})$. Output ρ

On random oracle query $\mathsf{KDF}_1(\rho, n_e)$ from $\mathcal{A}$:

K1.1 If no record $\langle \mathsf{KDF}_1, \rho, n_e, ([K_{\mathsf{export}}], [K_{\mathsf{mask}}], [K_{\mathsf{auth}}])\rangle$ exists:

(a) If records $\langle \text{PROG}, [\mathsf{kid}], \rho, [K_{\mathsf{auth}}]\rangle$, $\langle \text{INIT}, *, *, *, \mathsf{kid}, *, *, *, *, *, *, *, [x], [\mathsf{E}]\rangle$, and $\langle \mathsf{E}, * \parallel [\mathsf{e\_cred}] \parallel n_e \parallel *\rangle$ exist, compute $K_{\mathsf{export}} \xleftarrow{\$} \{0,1\}^\lambda$ and $K_{\mathsf{mask}} := \mathsf{e\_cred} \oplus x$ $(\mathbf{G}_{??})$

(b) Otherwise, choose $K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}} \xleftarrow{\$} \{0,1\}^\lambda$ $(\mathbf{G}_1)$

(c) If a record $\langle \mathsf{KDF}_1, *, *, (*, *, K_{\mathsf{auth}})\rangle$ exists, abort $(\mathbf{G}_4)$

(d) Record $\langle \mathsf{KDF}_1, \rho, n_e, (K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}})\rangle$ $(\mathbf{G}_1)$

K1.2 Output $(K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}})$ to $\mathcal{A}$ $(\mathbf{G}_1)$

On random oracle query $\mathsf{KDF}_2(\mathsf{ikm}, \mathsf{pre})$ from $\mathcal{A}$:

K2.1 If a record $\langle \mathsf{KDF}_2, \mathsf{ikm}, \mathsf{pre}, [s]\rangle$ exists, output s to $\mathcal{A}$ $(\mathbf{G}_1)$

K2.2 If records $\langle \text{DH}, [\mathsf{ssid}], \mathsf{ikm}, [s]\rangle$ and $\langle \text{RET}, \mathsf{ssid}, *, *, *, *, *, [m_1], *, [m_2], *, *, *, *\rangle$ exist such that $\mathsf{pre} = (a_2, n_C, U, Y, \mathsf{e\_cred}, n_e, T_e, b_2, n_S, V)$ for $m_1 = (n_C, U, a_2)$ and $m_2 = (b_2, \mathsf{e\_cred}, n_e, T_e, n_S, V, T_S, \sigma_2)$, record $\langle \mathsf{KDF}_2, \mathsf{ikm}, \mathsf{pre}, s\rangle$ $(\mathbf{G}_{13})$

K2.3 In any other case, send $(\mathsf{pre}, \mathsf{ikm})$ from $\mathcal{A}$ to $\text{SIM}_{\mathsf{3DH}}$. This triggers Step 5 of $\text{SIM}_{\mathsf{3DH}}$, which outputs s $(\mathbf{G}_{13})$

K2.4 Output s to $\mathcal{A}$. $(\mathbf{G}_1)$

Figure 4.23: Simulator $\text{SIM}_{\mathsf{WBP}}$ for WBP, part 6.

5 Password-Protected Key Retrieval with(out) HSM-Protection

Author's contribution. The contents of this chapter are based on joint work with Sebastian Faller, Julia Hesse, Máté Horváth, and Anja Lehmann [FHH+24]. The design of the three PPKR protocols presented in this chapter, which evolved through several mutual discussions, are an equal contribution of all authors. The majority of the security analysis of the OPRF-PPKR protocol given in Section 5.4 was developed by the author of this thesis with minor contributions from Sebastian Faller, Julia Hesse, and Máté Horváth in the form of occasional discussions of the proof strategy and the write up of small, isolated parts of the formal proof.

5.1 Overview

The question how users of IM apps can maintain access to their data when they lose their devices or switch to a new phone is one of the new challenges arising with the deployment of E2EE as the de facto security standard. Apart from WBP, which we analyzed extensively in Chapter 4, other protocols aiming at the secure backup of a user's chat history—or a user's cryptographic key in general—have been proposed by Signal [Lun19], Apple [Krs16], and Google [Wal18]. The approach of all three approaches is similar to the WBP in the sense that the user's high-entropy (encryption) key is stored on a server that relies on some trusted hardware enclaves, such as hardware security modules (HSMs). In our formal analysis of the WBP, we relied on the assumption that the HSM is perfectly secure and does not leak any cryptographic keys or backup files.

While this correctly captures the initial design choice made for the WBP protocol as well as the security claims advertised by WhatsApp [Wha21a], this is somewhat unsatisfactory from a security and protocol design perspective. First, if we assume an incorruptible HSM, the protocol design could take more advantage of that—the core of the WBP protocol is the OPAQUE protocol [JKX18], providing strong security guarantees even when the cryptographic state gets compromised, which wouldn't be needed if the assumption is that such an event can never occur. Second, relying on a perfectly secure sub-entity is a risky assumption. In fact, also trusted hardware modules have a history of getting breached or having to lower their security claims [VBMW+18, VBPS17, BC19, SRW22].

Therefore, in this chapter we explore the question how we can design highly efficient PPKR protocols under the assumption that the trusted hardware enclave

is incorruptible (Lev-1 security), partially corruptible in the sense that backup files may be leaked (Lev-2 security), or even fully corruptible (Lev-3 security).

5.1.1 Our Contributions

We propose three provably secure PPKR protocols, all relying on a server-internal HSM to protect the user's backup files, but aiming at a different trust level, thus having a dedicated trade-off between the strength of the HSM assumption and simplicity. We present two protocols that provide the same (or even stronger) *formally proven* security as the WBP, yet are significantly simpler. Both fall short of achieving security under full HSM corruption, for which we design a protocol that, similarly to the WBP, uses an oblivious pseudorandom function (OPRF) as a crucial building block, giving up on the reliance on basic primitives only. Let us provide some more detail.

We generally consider the same setting as for the WBP for all of our protocols. We assume client-to-server authenticated channels e.g. through an SMS or email passcode authentication as is done in WhatsApp, giving users the assurance that an adversary or other clients cannot try to retrieve their key. Importantly, we do not assume client-to-HSM authentication. In the other direction, as a notable difference from the WBP, we assume that the HSM authenticates all its messages through digital signatures, which is often referred to as attestation. Attestation is a standard feature of HSMs and must be deployed with some means of verifying the authenticity of the signature public key, e.g. with a certificate chain.[1] The details for this, however, are beyond the scope of this thesis. As in Chapter 4, we model the HSM as a hybrid functionality providing an interface where any party can obtain the public key, reflecting that there exists some trusted way of obtaining the key in practice.

In our model of leakage from the HSM, covered through Lev-2 security, HSM attestation is the *only* power kept from the adversary. Only in our strongest model, Lev-3 that considers full HSM corruption, we will provide the adversary with the attestation key. This distinction between the permanent attestation key and client-specific files is justified because attestation is a central capability of HSMs. Therefore, an HSM manufacturer might turn special attention to the protection of the single attestation key, such as more expensive hardware protection mechanisms.

For simplicity, we also assume that the HSM has some persistent memory where it stores file records. In practice, HSMs might securely outsource such storage as done, e.g., in the case of the WBP (cf. Section 4.2.4), but the actual realization of storage is irrelevant for our work. This HSM-protected storage is still considered secret upon server corruption (Lev-1) but entirely leaked to the adversary from Lev-2 up.

[1] See e.g. `https://thalesdocs.com/gphsm/luna/7/docs/network/Content/admin_partition/confirm/confirm_hsm.htm`

Finally, we provide an intuition for the basic approaches of our three constructions.

Basic/enhanced encrypt-to-HSM: We start with a protocol that fully relies on a trusted HSM for most of its operation, and leverage this to simplify the design as much as possible. The resulting protocol merely uses standard symmetric and asymmetric encryption. We prove our simple protocol to be secure on Lev-1, which gives the same (or even slightly stronger, see discussions in Sections 4.2.6 and 4.3.3) security guarantees as we have formally proven for WBP in Section 4.4.3. We further show how to upgrade the protocol to Lev-2 security, by relying on fresh encryption keys and salted hashing in the stored password files. The simplicity hinges on the trust in the HSM though, as the protocol loses its security if the HSM gets fully corrupted. Interestingly, the best attacks against all long-term user files are still offline attacks, but the login passwords that users send during an active retrieval session are now revealed in plain when the HSM is fully corrupted, which violates the Lev-3 guarantees.

OPRF-based: Our third protocol provides Lev-3 security, i.e., it guarantees optimal protection even in the presence of full server and HSM corruption. This protocol partly resembles the WBP, as it also relies on an OPRF as a core primitive. Recall that the WBP relies on OPAQUE, an aPAKE that is built from an OPRF. By building our protocol directly from OPRFs and the clearly specified security guarantees as concrete target, our protocol enjoys a much cleaner design. In particular, we can omit all OPAQUE parts that are not needed for PPKR (which does not aim at fresh session keys as aPAKE). Overall, we propose an OPRF-based PPKR that has a simpler protocol design and better efficiency than WBP, and that fixes several attacks related to user authentication that we identified in Section 4.2.6.

For an overview of the three protocols, see Table 5.1.

5.2 Lev-1 Protocol: Basic Encrypt-to-HSM

In the security analysis of the WBP in Section 4.4.3 we assumed that the HSM cannot be corrupted and never leaks information. A natural question is if this strong assumption allows for a simpler PPKR protocol. The basic encrypt-to-HSM protocol encPw, depicted in Figure 5.1, answers the question in the affirmative.

The central observation is that if no information is leaked by the HSM, then one can store the client's passwords and keys in the clear at the HSM.When a client wants to retrieve its key, the HSM can simply compare the provided password with the stored cleartext password and return the key if the check was successful. We must only ensure that the password and the key remain protected during transmission via the (possibly corrupt) server—the prototypical scenario

Table 5.1: Security achieved by our PPKR protocols and WBP. ✓= achieves optimal protection in that corruption setting, ✗= not secure, where Lev-1: server corrupt, Lev-2: server corrupt + backup file leaks, Lev-3: server (and HSM) fully corrupt. Gray are conjectures. WBP was proven secure in a model that is slightly weaker than our Lev-1 model, * denotes that similar relaxations likely being needed for the given level too. The last column indicates if the protocol relies on standard building blocks only.

	Lev-1	Lev-2	Lev-3	Standard BB
encPw, Figure 5.1	✓	✗	✗	✓
encPw+, Figure 5.5	✓	✓	✗	✓
OPRF-PPKR, Figure 5.9	✓	✓	✓	✗
WBP, Figures 4.4 and 4.5	✓*	✓*	✓*	✗

for encryption. Therefore the core idea behind the pw-to-HSM protocol is as follows:

1. For initialization, the client encrypts its password and its randomly sampled backup key K under the public key of the HSM: $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, (\mathsf{pw}, K))$. The client sends C via the server to the HSM.

2. The HSM decrypts C and stores the password and the backup key.

3. For recovery, the client encrypts its password together with a freshly chosen symmetric key $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, (\mathsf{pw}', k))$ and sends that to the HSM.

4. The HSM decrypts C and compares pw' to the stored password pw. If the check is successful, the HSM encrypts the stored backup key under the provided symmetric key $C' \xleftarrow{\$} \mathsf{SE.Enc}(k, K)$. The HSM sends the C' to the client.

5. The client decrypts the symmetric ciphertext C' and obtains its backup key K.

We assume that the public key pk can be distributed similarly as the public attestation key and thus all clients have access to pk. However, there are a number of subtleties that have to be addressed by the protocol in order to satisfy $\mathcal{F}_{\mathsf{PPKR}}$.

To prevent the "rerouting" of messages to a different cid by a corrupt server, we let the client encrypt session identifier ssid and their own identity cid, and also include these items in the clear. By comparing the decrypted identifiers with the clear-text ones, the HSM will notice when a malicious server changes the identifiers of honest client requests, and abort. Hence, with this little check, encPw is stronger (on Lev-1) than the WBP, which admits such attacks as discussed in Section 4.2.6.

This means we can even prove the basic encrypt-to-HSM protocol is secure under the strengthened version of $\mathcal{F}_{\mathsf{PPKR}}$, where we drop the code in dashed boxes (cf. Section 4.3.3). Moreover, as the HSM directly checks the password supplied by the client, it is not possible for corrupt parties to reset their counter when retrieving with a wrong password and we can also drop the code in solid boxes of $\mathcal{F}_{\mathsf{PPKR}}$.

We show that encPw is a secure PPKR protocol when only considering server corruptions, i.e., it provides Lev-1 security.

Theorem 6. *Let* $\mathsf{SE} = (\mathsf{SE.KGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ *be an* IND-CPA*-secure symmetric encryption scheme,* $\mathsf{PKE} = (\mathsf{PKE.KGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an* IND-CCA*-secure public key encryption scheme, and* $\mathsf{Sig} = (\mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Vrfy})$ *by an* sEUF-CMA*-secure signature scheme used for attestation.*

Then, the protocol encPw *from Figure 5.1 UC-realizes* $\mathcal{F}_{\mathsf{PPKR}}$ *at* Lev-1 *security (i.e. without the* LeakFile *and* FullyCorrupt *interfaces) in the* $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$*-hybrid model, malicious adaptive corruptions , and a client-authenticated channel between clients and the server. Concretely, we can construct adversaries* $\mathcal{B}_1, \ldots \mathcal{B}_4$ *such that for any efficient adversary against* encPw *(interacting with* $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$*), the simulator* $\mathrm{SIM}_{\mathsf{encPw}}$ *interacting with* $\mathcal{F}_{\mathsf{PPKR}}$ *produces a view such that for every efficient environment* $\mathcal{Z}$ *it holds that*

$$\begin{aligned}\mathbf{Adv}^{\mathcal{F}_{\mathsf{PPKR}}}_{\mathsf{encPw},\mathrm{SIM}_{\mathsf{encPw}},\mathcal{Z}}(\lambda) \leq & \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda) + q_{\mathrm{INIT}}\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_2}(\lambda) \\ & + q_{\mathrm{RET}}\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_3}(\lambda) + q_{\mathrm{RET}}\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{B}_4}(\lambda),\end{aligned}$$

where q_{INIT} *is the number of initializations and* q_{RET} *is the number of retrievals.*

Proof intuition. The gist of the proof is that the incorruptible HSM holds the secret key $\mathsf{sk}_{\mathsf{enc}}$ for all encryptions. As the HSM is modeled as a hybrid functionality, the simulator has access to the secret key. This allows for the following proof strategy:

To simulate *honest clients* without the password, the simulator can replace the PKE ciphertexts C by encryptions of 0-strings of appropriate length. The environment cannot detect this change by the IND-CCA security of the encryption scheme. Indeed, we require CCA security, as the HSM's responses on mismatching cid and ssid give the adversary a very limited decryption oracle. The ciphertext C' is again replaced by an encryption of 0-strings of appropriate length. Note that this time we only need to reduce to IND-CPA security because the integrity of C' is ensured by the HSM attestation. Finally, the simulator can use $\mathcal{F}_{\mathsf{PPKR}}$ to provide clients with their correct output.

To extract the inputs of *corrupt clients* the simulator can use the secret key $\mathsf{sk}_{\mathsf{enc}}$ of the HSM. The simulator knows this key because the HSM is modeled as a hybrid functionality. That means concretely that in the ideal-world execution, the simulator plays the role of the HSM (and thus, chooses $\mathsf{sk}_{\mathsf{enc}}$ by itself). Using $\mathsf{sk}_{\mathsf{enc}}$, the simulator can decrypt the PKE ciphertext C provided by the corrupt client and provide the used password pw^* to $\mathcal{F}_{\mathsf{PPKR}}$. If the password guess was

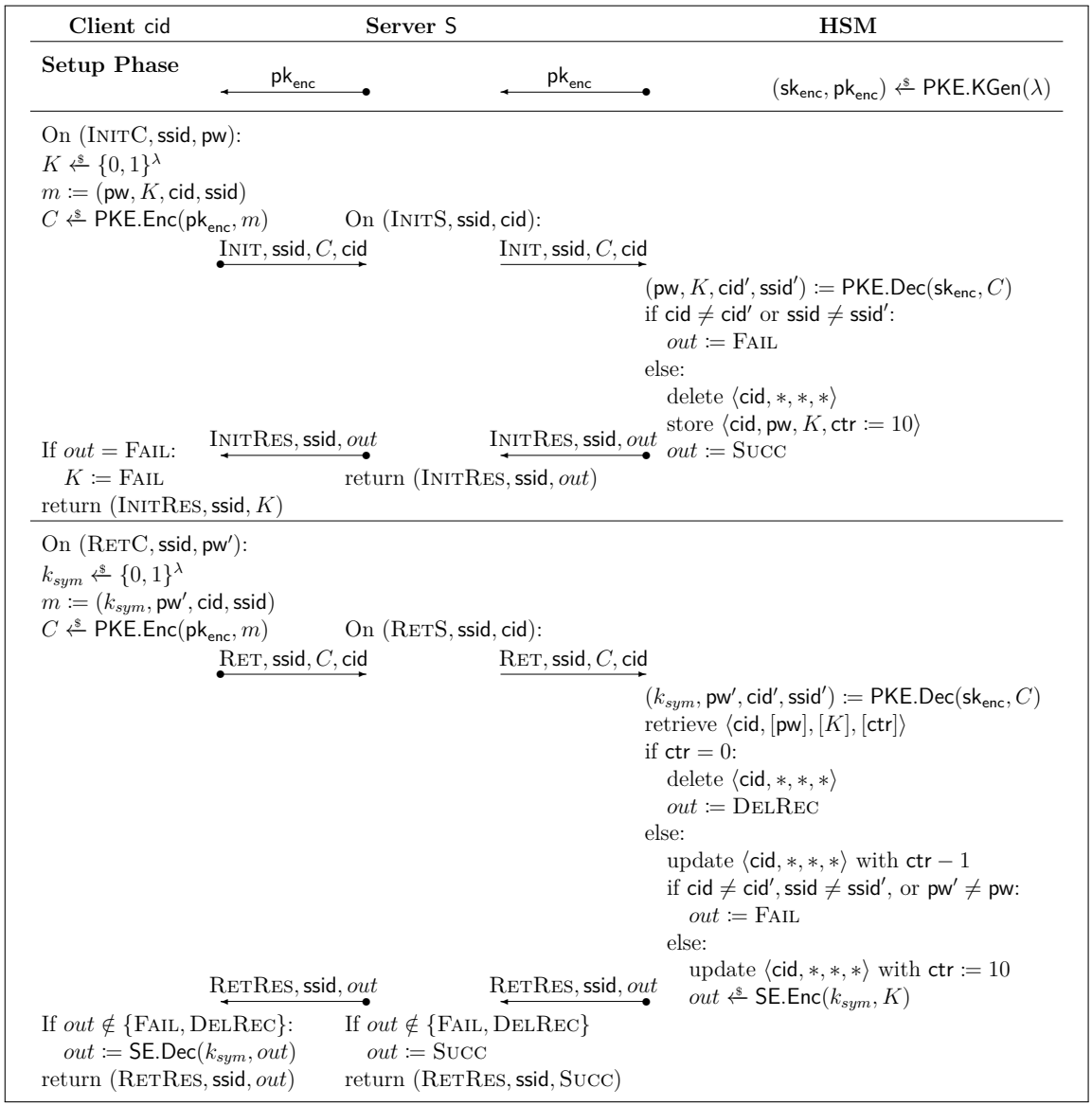

Figure 5.1: Protocol encPw. $\xleftarrow{x}$ is the HSM-attested transmission of x. $\xrightarrow{y}$ is the client-to-server authenticated transmission of y. We implicitly assume that each message contains sid. cid and S output (InitRes, ssid, Fail) or (RetRes, ssid, Fail), whenever they receive an unexpected message (i.e. with mismatching ssid or cid) or when attestation verification fails. If any party receives the same type of message twice with the same ssid, it ignores the second one.

Functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$

The functionality initially computes $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and stores $\langle \mathsf{sk}_{\mathsf{enc}} \rangle$.

On $(\textsc{Init}, \mathsf{ssid}, C, \mathsf{cid})$ from S:

- Retrieve $\langle [\mathsf{sk}_{\mathsf{enc}}] \rangle$
- $(\mathsf{pw}, K, \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$
- If $\mathsf{cid}' \neq \mathsf{cid}$ or $\mathsf{ssid}' \neq \mathsf{ssid}$, set $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Fail})$ to S. Else continue.
- Delete $\langle \mathsf{cid}, *, *, * \rangle$ and store $\langle \mathsf{cid}, \mathsf{pw}, K, 10 \rangle$
- Send $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Succ})$

On $(\textsc{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ from S:

- Retrieve $\langle \mathsf{sk}_{\mathsf{enc}} \rangle$.
- $(k_{sym}, \mathsf{pw}', \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk}, C)$
- Retrieve $\langle \mathsf{cid}, [\mathsf{pw}], K, [\mathsf{ctr}] \rangle$
- If $\mathsf{ctr} = 0$, delete $\langle \mathsf{cid}, \mathsf{pw}, K, \mathsf{ctr} \rangle$ and send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{DelRec})$ to S. Else continue.
- If $\mathsf{cid}' \neq \mathsf{cid}, \mathsf{ssid}' \neq \mathsf{ssid}$, or $\mathsf{pw} \neq \mathsf{pw}'$, update $\langle \mathsf{cid}, *, *, * \rangle$ with $\mathsf{ctr} - 1$ and send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ to S. Else continue.
- Update $\langle \mathsf{cid}, *, *, * \rangle$ with $\mathsf{ctr} := 10$
- $C' \xleftarrow{\$} \mathsf{SE.Enc}(k_{sym}, K)$. Set $r := C'$.
- Send $(\textsc{RetRes}, \mathsf{ssid}, C')$ to S

Figure 5.2: The ideal functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$.

correct the functionality will give the simulator the retrieved backup key K. The simulator can also extract the corrupt client's symmetric key k_{sym} by using the secret key $\mathsf{sk}_{\mathsf{enc}}$ of the HSM and respond to the client with an encrypted version of its backup key C'.

Simulating the server amounts to executing the rest of the protocol as usual because the server holds no private information.

Proof. We construct a sequence of hybrid games $\mathbf{G}_0$ to $\mathbf{G}_{14}$ where we gradually change the real-world execution of the protocol encPw (interacting with the hybrid functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$) to reach the ideal-world execution, where the environment interacts with the simulator from Figures 5.3 to 5.4 and the ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$. We write $\Pr[\mathbf{G}_i]$ to denote the probability that the environment outputs 1 in the hybrid game $\mathbf{G}_i$.

Game $\mathbf{G}_0$: Real world. This is the real world.

Game $\mathbf{G}_1$: Create simulator. In this game we create two new entities called the ideal functionality $\mathcal{F}$ and the simulator Sim. Initially, $\mathcal{F}$ just forwards the input of the dummy parties to Sim and outputs what Sim instructs it to output. In particular, $\mathcal{F}$ has interfaces $(\textsc{InitC}, \mathsf{ssid}, \mathsf{pw})$, $(\textsc{InitS}, \mathsf{ssid}, \mathsf{cid})$,$(\textsc{RetC}, \mathsf{ssid}, \mathsf{pw}')$, and $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}^*, K^*, i)$ that just forward the input to Sim. The simulator executes the code of all honest parties of the protocol internally on the input that it is provided by $\mathcal{F}$ and it internally runs the code of the hybrid functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$. Note that these are just syntactical changes and the protocol is still executed as in the real world. We have

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Game $\mathbf{G}_2$: Output Fail on tampered message from HSM. In this game, we change how the simulator reacts when an honest client receives a message from the HSM that was tampered with. Whenever an honest client receives a message $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Succ})$ where Sim never produced this message on behalf of Sim, then Sim makes the client output Fail (see IO.1). Similarly, when an honest client receives a message $(\textsc{RetRes}, \mathsf{ssid}, \mathit{out})$ that was never produced by Sim on behalf of the HSM, then Sim makes the client output Fail (see RO.1).

The distribution of $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical, unless some honest client receives a message that was never simulated by Sim on behalf of the HSM, but where the attestation signature is still valid, which we denote as the event $\mathrm{E}_{\mathsf{Sig}}$. Assume there is some environment $\mathcal{Z}^*$ that is able to distinguish between $\mathbf{G}_2$ and $\mathbf{G}_1$. Then we construct an adversary $\mathcal{B}_1$ against the $\mathsf{sEUF\text{-}CMA}$-security of Sig. $\mathcal{B}_1$ internally runs the whole experiment including $\mathcal{F}$, Sim and $\mathcal{Z}^*$. Whenever it simulates a message on behalf of the HSM it uses the signature oracle Sign to compute the attestation signature. This

Initially, compute $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}}) := \mathsf{PKE.KGen}(1^\lambda)$ and store record $\langle \mathsf{sid}, \mathsf{sk_{enc}}, \mathsf{pk_{enc}} \rangle$. *Wait for* X means that $\text{SIM}_{\mathsf{encPw}}$ does not proceed to the next instruction before receiving X and meanwhile gives back activation to $\mathcal{A}$. Once it receives X, it first proceeds with the instructions on input X and then continues at the instruction, where it waited for X. If a record cannot be retrieved, the query is ignored. For brevity we omit session identifier sid from all inputs, outputs, and records.

On $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

I.1 Retrieve record $\langle \mathsf{sid}, *, [\mathsf{pk_{enc}}] \rangle$ $(\mathbf{G}_1)$

I.2 Compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk_{enc}}, m)$, where m is a 0-string of appropriate length, and record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, C \rangle$ $(\mathbf{G}_3)$. Send $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ to S on behalf of cid $(\mathbf{G}_1)$.

On $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

I.3 If S is honest, wait for $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from cid to S. $(\mathbf{G}_1)$

On $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of cid:

IC.1 Wait for $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_1)$

IC.2 If cid is honest, retrieve $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, C \rangle$. $(\mathbf{G}_3)$

IC.3 If cid is corrupt:

a) Retrieve $\langle \mathsf{sid}, [\mathsf{sk_{enc}}], * \rangle$, compute $(\mathsf{pw}, K, \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk_{enc}}, C)$ $(\mathbf{G}_1)$ and give input $(\text{INITC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of cid, where $\mathsf{pw} = \perp$ if the decryption resulted in a 0-string. On response $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ continue below. $(\mathbf{G}_5)$

b) If the decryption resulted in a 0-string $(\mathbf{G}_3)$, $\mathsf{cid} \neq \mathsf{cid}'$, or $\mathsf{ssid} \neq \mathsf{ssid}'$, send $(\text{INITRES}, \mathsf{ssid}, \text{FAIL})$ to cid $(\mathbf{G}_1)$ and $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_6)$ // 0 meaning fail.

c) Otherwise, store $\langle \text{FILE}, \mathsf{cid}, K \rangle$, overwriting any existing $\langle \text{FILE}, \mathsf{cid}, * \rangle$. $(\mathbf{G}_1)$ // No need to store pw and ctr, as $\mathcal{F}_{\mathsf{PPKR}}$ takes care of this.

IC.4 If a record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, C \rangle$ exists, store record $\langle \text{FILE}, \mathsf{cid}, \mathsf{ctr} := 10 \rangle$. $(\mathbf{G}_1)$ // SIM must keep counter for DELREC in honest IDC, honest server case.

IC.5 Send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to cid $(\mathbf{G}_1)$ and give input $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_6)$ // 1 meaning no fail.

On $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of corrupt S:

IC.6 If there is a record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, C \rangle$ $(\mathbf{G}_3)$, then give input $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_6)$. // Honest C from cid is used
On response $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$, send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to S $(\mathbf{G}_3)$. Send $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_6)$. // $b_S = 1$ for successful Init.

IC.7 Else if there is no such record, then retrieve record $\langle \mathsf{sid}, [\mathsf{sk_{enc}}], * \rangle$ and compute $(\mathsf{pw}^*, K^*, \mathsf{cid}^*, \mathsf{ssid}^*) := \mathsf{Dec}(\mathsf{sk_{enc}}, C)$ $(\mathbf{G}_1)$. If the decryption results in a 0-string $(\mathbf{G}_3)$, $\mathsf{cid}^* \neq \mathsf{cid}$, or $\mathsf{ssid}^* \neq \mathsf{ssid}$, then return $(\text{RETRES}, \mathsf{ssid}, \text{FAIL})$ to S $(\mathbf{G}_1)$. Else give input $(\text{MALICIOUSINIT}, \mathsf{cid}, \mathsf{pw}^*, K^*)$ to $\mathcal{F}_{\mathsf{HSM}}$ $(\mathbf{G}_4)$, delete any existing record $\langle \text{FILE}, \mathsf{cid}, * \rangle$ $(\mathbf{G}_{12})$, and send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to S $(\mathbf{G}_1)$. // Maliciously chosen key and pwd.

On $(\text{INITRES}, \mathsf{ssid}, \mathit{out})$ from $\mathcal{A}$ to cid on behalf of S:

IO.1 If $(\text{INITRES}, \mathsf{ssid}, \mathit{out})$ was never output by $\text{SIM}_{\mathsf{encPw}}$ on behalf of S or $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, send $(\text{COMPLETEINITC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_2)$. Else if there is no record $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, * \rangle$, give input $(\text{COMPLETEINITC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{11})$. // 0 meaning fail.

IO.2 Else, give input $(\text{COMPLETEINITC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{11})$ // 1 meaning no success.

Figure 5.3: The initialization part of the simulator $\text{SIM}_{\mathsf{encPw}}$ for the protocol encPw.

implicitly programs the attestation key of the HSM to the one sampled in the $\mathsf{sEUF\text{-}CMA}$ experiment. As the key is sampled uniformly at random both in $\mathsf{sEUF\text{-}CMA}$ experiment and $\mathbf{G}_2$, this does not change the distribution of the key. When the event $\mathrm{E}_{\mathsf{Sig}}$ occurs, $\mathcal{B}_1$ outputs the corresponding message and signature to its challenger. It is easy to see that $\mathcal{B}_1$ wins the experiment whenever $\mathrm{E}_{\mathsf{Sig}}$ occurs. Hence, we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda).$$

Game $\mathbf{G}_3$: Switch ciphertext in initialization. In this game, we change how Sim computes the ciphertext C produced by honest clients during initialization. Instead of computing $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk_{enc}}, (\mathsf{pw}, K, \mathsf{cid}, \mathsf{ssid}))$, the simulator computes $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk_{enc}}, m)$, where m is a 0-string of appropriate length. Nonetheless, Sim still receives the secret input pw from $\mathcal{F}$ and chooses $K := \{0,1\}^\lambda$, which it stores in a record $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, C, \mathsf{pw}, K\rangle$ (see I.2). Note that storing pw and K in the record is only a temporary change and we remove this again in $\mathbf{G}_{11}$.

Further, we change how Sim acts when receiving a ciphertext C from an honest cid or from a corrupt server that honestly delivers C from an honest cid to $\mathcal{F}^{\mathsf{encPw}}_{\mathsf{HSM}}$ in an initialization. In more detail, Sim keeps track of all ciphertexts that it computes for honest cid as $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk_{enc}}, m)$ in the $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, C, \mathsf{pw}, K\rangle$ records. When Sim receives a message $(\textsc{Init}, \mathsf{ssid}, C, \mathsf{cid})$, then Sim tries to retrieve $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, C, [\mathsf{pw}], [K]\rangle$, i.e., it checks if C was computed by Sim for the honest client cid as an encryption of a 0-string in subsession ssid. If that is the case, then Sim does not use $\mathsf{sk_{enc}}$ to decrypt C but instead stores a record $\langle \textsc{File}, \mathsf{cid}, \mathsf{pw}, K, 10\rangle$ and sends $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Succ})$ to the server (see IC.2 and IC.6). Again, creating this record is only a temporary change that we remove again in games $\mathbf{G}_9$, $\mathbf{G}_{11}$ and $\mathbf{G}_{14}$. If C was not computed by Sim, indicated by the fact that no record $\langle \textsc{Init}, \mathsf{ssid}, \mathsf{cid}, C, *, *\rangle$ exists, then Sim continues as in $\mathbf{G}_2$. Note that this may lead to Sim decrypting C to a 0-string, e.g., if $\mathcal{A}$ replays some C that was computed by Sim on behalf of some honest cid. However, in that case, in $\mathbf{G}_2$, Sim would output Fail to S as there would be a mismatch between cid and cid' or ssid and ssid'. Thus, we let Sim output Fail to S if C is decrypted to a 0-string (see IC.3 (b) and IC.7).

It is easy to see that the outputs of the server and the client are just as in $\mathbf{G}_2$. Hence, the only difference is the distribution of C. If $\mathcal{Z}$ can distinguish $\mathbf{G}_3$ from $\mathbf{G}_2$, then we can construct an adversary $\mathcal{B}_1$ against the $\mathsf{IND\text{-}CCA}$ security of PKE as follows: First, $\mathcal{B}_1$ does not compute $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}})$ itself but uses the pk^* provided by its challenger. Let $q_{\textsc{Init}} \in \mathbb{N}$ be the number of initializations. We construct a sequence of games $\mathbf{G}_2^{(0)}, \ldots, \mathbf{G}_2^{(q_{\textsc{Init}})}$, where in $\mathbf{G}_2^{(i)}$ the first i ciphertexts are computed as encryptions of 0-strings if simulated for an honest cid and the remaining ciphertexts are encrypted as in $\mathbf{G}_2$

(except that pk^* is used). We have $\mathbf{G}_2 = \mathbf{G}_2^{(0)}$ and $\mathbf{G}_3 = \mathbf{G}_2^{(q_{\text{INIT}})}$. Because $\mathcal{Z}$ can distinguish $\mathbf{G}_2$ from $\mathbf{G}_3$, there must be an index $i^* \in [q_{\text{INIT}}]$ such that $\mathcal{Z}$ has a non-negligible advantage in distinguishing $\mathbf{G}_2^{(i^*)}$ and $\mathbf{G}_2^{(i^*-1)}$. Now, in the i^*-th initialization the reduction $\mathcal{B}_1$ gives $m_0 := (k_{sym}, \mathsf{pw}, \mathsf{cid}, \mathsf{ssid})$ and m_1 a 0-string of the same length to the challenger and uses the returned C^* as ciphertext for the i^*-th initialization. Additionally, whenever $\mathcal{B}_1$ receives a message $(\text{INIT}, \mathsf{ssid}, C^*, \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of a corrupt server such that there is no record $\langle \text{INIT}, \mathsf{ssid}, C^*, \mathsf{cid}, [\mathsf{pw}], [K] \rangle$, then $\mathcal{B}_1$ uses its decryption oracle to decrypt C^*. The oracle answers with $(\mathsf{pw}^*, K^*, \mathsf{cid}^*, \mathsf{ssid}^*)$ and $\mathcal{B}_1$ then proceeds as in $\mathbf{G}_2$.

Now, if C^* encrypts m_0, the game is distributed as in $\mathbf{G}_2^{(i^*-1)}$ and if it encrypts m_1, then the game is distributed as in $\mathbf{G}_2^{(i^*)}$. We get

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq q_{\text{INIT}} \mathbf{Adv}_{\mathsf{PKE},\mathcal{B}_1}^{\mathsf{IND\text{-}CCA}}(\lambda).$$

Game $\mathbf{G}_4$: Inform $\mathcal{F}$ of malicious initializations. In this game, we add the MALICIOUSINIT interface as in $\mathcal{F}_{\mathsf{PPKR}}$ to $\mathcal{F}$. We also let SIM use MALICIOUSINIT to make $\mathcal{F}$ create records for malicious initializations. More precisely, when $\mathcal{A}$ instructs a corrupted server to send a message $(\text{INITC}, \mathsf{ssid}, C^*, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, where C^* was never sent by an honest client before, then SIM first proceeds just as in $\mathbf{G}_3$, i.e., it executes the code of $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, but additionally gives input $(\text{MALICIOUSINIT}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}^*, K^*)$ to $\mathcal{F}$ (see IC.7).

Note that on receiving the MALICIOUSINIT input $\mathcal{F}$ only responds to SIM. Therefore, $\mathcal{Z}$'s view did not change and we get

$$\Pr[\mathbf{G}_4] = \Pr[\mathbf{G}_3].$$

Game $\mathbf{G}_5$: Provide $\mathcal{F}$ with input of corrupted clients in initialization. In this game we change SIM such that when it receives a message $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from a corrupted cid to an honest S, in addition to decrypting $(\mathsf{pw}, K, \mathsf{cid}', \mathsf{ssid}')$, checking $\mathsf{cid} = \mathsf{cid}', \mathsf{ssid} = \mathsf{ssid}'$ and recording $\langle \text{FILE}, \mathsf{cid}, \mathsf{pw}, K, 10 \rangle$, it also gives input $(\text{INITC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}$ (see IC.3 (a)). Since a corrupted cid may replay ciphertexts C that were computed by SIM as an encryption of a 0-string, SIM sets $\mathsf{pw} = \perp$ for its INITC query if C decrypts to a 0-string. This cannot modify the view of $\mathcal{Z}$ as SIM outputs FAIL to the corrupt client if such a replay happens (cf. IC.3 (b)). Note that the INITC interface of $\mathcal{F}$ is still a dummy interface, so this change does not alter the view of $\mathcal{Z}$, so

$$\Pr[\mathbf{G}_5] = \Pr[\mathbf{G}_4].$$

Game $\mathbf{G}_6$: Let $\mathcal{F}$ generate server output in initialization. In this game, we change the initialization interfaces INITC and INITS, and add the interface

On $(\textsc{RetC}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

R.1 Retrieve record $\langle \mathsf{sid}, [\mathsf{sk_{enc}}], [\mathsf{pk_{enc}}] \rangle$. $(\mathbf{G}_1)$

R.2 Choose $k_{sym} \xleftarrow{\$} \{0,1\}^\lambda$ $(\mathbf{G}_1)$. Compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk_{enc}}, m)$, where m is a 0-string of appropriate length and record $\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, C, k_{sym} \rangle$ $(\mathbf{G}_7)$. Send $(\textsc{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ to S on behalf of cid. $(\mathbf{G}_1)$

On $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}', \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

R.3 If S is honest, wait for $(\textsc{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ from cid to S. $(\mathbf{G}_1)$

On $(\textsc{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of cid:

RC.1 Wait for $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_1)$

RC.2 If cid is honest:

a) Retrieve $\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, C, [k_{sym}] \rangle$. $(\mathbf{G}_7)$

b) Retrieve record $\langle \textsc{File}, \mathsf{cid}, [\mathsf{ctr}] \rangle$. If $\mathsf{ctr} = 0$ set $C' := \textsc{DelRec}$. Else if $\mathit{match} = 1$ $(\mathbf{G}_{14})$, compute $C' \xleftarrow{\$} \mathsf{SE.Enc}(k_{sym}, 0^\lambda)$ $(\mathbf{G}_{10})$ and set ctr in the record to 10. Else set $C' := \textsc{Fail}$ $(\mathbf{G}_1)$ and ctr in the record to $\mathsf{ctr} - 1$.

c) Store $\langle \mathsf{ssid}, C' \rangle$ $(\mathbf{G}_7)$.

RC.3 If cid is corrupt:

a) Retrieve $\langle \mathsf{sid}, [\mathsf{sk_{enc}}], * \rangle$, compute $(k_{sym}, \mathsf{pw}', \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk_{enc}}, C)$ $(\mathbf{G}_1)$, and give input $(\textsc{RetC}, \mathsf{ssid}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of cid, where $\mathsf{pw}' = \perp$ if the decryption resulted in a 0-string. On response $(\textsc{RetC}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ continue below. $(\mathbf{G}_8)$

b) If the decryption resulted in a 0-string $(\mathbf{G}_7)$, $\mathsf{cid} \neq \mathsf{cid}'$, or $\mathsf{ssid} \neq \mathsf{ssid}'$, send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ to cid. $(\mathbf{G}_1)$

c) Send $(\textsc{CompleteRetC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetRes}, \mathsf{ssid}, K)$, if $K \in \{\textsc{Fail}, \textsc{DelRec}\}$, set $C' := K$. $(\mathbf{G}_{12})$

d) If $K \notin \{\textsc{Fail}, \textsc{DelRec}\}$ and a record $\langle \textsc{File}, \mathsf{cid}, [K'] \rangle$ exists, compute $C' \xleftarrow{\$} \mathsf{SE.Enc}(k'_{sym}, K')$ $(\mathbf{G}_{12})$. // Ignore, K, it is different from the extracted K'

e) If no such record exists, compute $C' \xleftarrow{\$} \mathsf{SE.Enc}(k'_{sym}, K)$. $(\mathbf{G}_1)$ // cid was honest at Init

RC.4 Send $(\textsc{CompleteRetS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_9)$ and $(\textsc{RetRes}, \mathsf{ssid}, C')$ to cid. $(\mathbf{G}_1)$

On $(\textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, C)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of corrupt S:

RC.5 If a record $\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, C, [k_{sym}] \rangle$ exists, then give input $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}, \perp, \perp, \perp)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$, give input $(\textsc{CompleteRetS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_9)$ and on the subsequent response $(\textsc{RetRes}, \mathsf{ssid}, K')$ execute the steps RC.2 (b) and RC.2 (c). $(\mathbf{G}_{14})$

RC.6 If no record $\langle \textsc{Ret}, \mathsf{ssid}, \mathsf{cid}, C, * \rangle$ exists, then retrieve $\langle \mathsf{sid}, \mathsf{sk_{enc}}, \mathsf{pk_{enc}} \rangle$ and compute $(k_{sym}, \mathsf{pw}', \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk_{enc}}, C)$. $(\mathbf{G}_1)$

RC.7 If the decryption results in a 0-string $(\mathbf{G}_7)$, $\mathsf{cid}' \neq \mathsf{cid}$, or $\mathsf{ssid}' \neq \mathsf{ssid}$, then send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ to S. $(\mathbf{G}_1)$

RC.8 Give input $(\textsc{MaliciousRet}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PPKR}}$ and on response K' determine the output as follows $(\mathbf{G}_{12})$.

a) If $K' \in \{\textsc{DelRec}, \textsc{Fail}\}$, then send $(\textsc{RetRes}, \mathsf{ssid}, K')$ to S. $(\mathbf{G}_{12})$

b) If $K' \notin \{\textsc{DelRec}, \textsc{Fail}\}$, then compute $C' := \mathsf{SE.Enc}(k_{sym}, K)$ if a record $\langle \textsc{File}, \mathsf{cid}, [K] \rangle$ exists and $C' := \mathsf{SE.Enc}(k_{sym}, K')$ otherwise. Send $(\textsc{RetRes}, \mathsf{ssid}, C')$ to S $(\mathbf{G}_{12})$.

On $(\textsc{RetRes}, \mathsf{ssid}, \mathit{out})$ from $\mathcal{A}$ to cid on behalf of S:

RO.1 If HSM signature does not verify, send $(\textsc{CompleteInitC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_2)$. Else if $\mathit{out} \in \{\textsc{Fail}, \textsc{DelRec}\}$, give input $(\textsc{CompleteRetC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{11})$ // If it is DelRec then the DoS bit doesn't matter.

RO.2 Else give input $(\textsc{CompleteRetC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{11})$ // $\mathit{out} = C'$ is attestated by the HSM and $\mathcal{A}$ didn't tamper with it.

Figure 5.4: The retrieval part of the simulator $\textsc{Sim}_{\mathsf{encPw}}$ for the protocol encPw.

CompleteInitS to $\mathcal{F}$. We change InitC and InitS to be exactly as in $\mathcal{F}_{\mathsf{PPKR}}$ except that InitC still provides Sim with the input pw of cid. Further, we change Sim such that it uses CompleteInitS to produce output for honest S in the initialization phase. Now, when Sim receives $(\text{InitS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}$ and a message $(\text{InitC}, \mathsf{ssid}, C, \mathsf{cid})$ to the server, where either C was produced by Sim itself or $(\mathsf{pw}, K, \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$ and $\mathsf{cid}' = \mathsf{cid}$ and $\mathsf{ssid}' = \mathsf{ssid}$, then Sim does not directly make the server output $(\text{InitRes}, \mathsf{ssid}, \mathsf{cid}, \text{Succ})$ but instead gives $(\text{CompleteInitS}, \mathsf{ssid}, \mathsf{cid}, 1)$ to $\mathcal{F}$ (IC.5). If the decrypted cid' is different from cid or ssid' is different from ssid, then Sim gives input $(\text{CompleteInitS}, \mathsf{ssid}, \mathsf{cid}, 0)$ to $\mathcal{F}$ (IC.3 (b)).

Furthermore, Sim uses the CompleteInitS interface, whenever a corrupt S honestly delivers a message from some honest cid to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$. Even though in that case the interface gives its output to Sim, this query is necessary to ensure that $\mathcal{F}$ creates a File record. Note that $\mathcal{F}$ only creates a File record, if it received the InitC and InitS input. Thus, since S is corrupt, Sim first gives the input $(\text{InitS}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{F}$ (see IC.6).

We can see that the output of the honest server did not change. In $\mathbf{G}_5$ the honest server outputs $(\text{InitRes}, \mathsf{ssid}, \text{Succ})$ if it received a ciphertext that decrypted to the correct $\mathsf{cid}, \mathsf{ssid}$. If C contains a different cid' or ssid', the server aborts. The simulator makes $\mathcal{F}$ provide the corresponding output by choosing the bit b_S accordingly for the CompleteInitS message. Hence,

$$\Pr[\mathbf{G}_6] = \Pr[\mathbf{G}_5].$$

Note that due to the changes introduced in Games $\mathbf{G}_4$-$\mathbf{G}_6$, $\mathcal{F}$ now stores File records in all initializations, although $\mathcal{F}$ does not use them in retrievals, yet.

Game $\mathbf{G}_7$: Switch ciphertext in retrieval. In this game, we change how Sim computes the ciphertext C produced by honest clients during retrieval. Instead of computing the ciphertext as $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, (k_{sym}, \mathsf{pw}', \mathsf{cid}, \mathsf{ssid}))$ the simulator computes it as $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, m)$, where m is a 0-string of appropriate length. Still, it chooses a symmetric key $k_{sym} \xleftarrow{\$} \{0,1\}^\lambda$ and stores a record $\langle \text{Ret}, \mathsf{ssid}, \mathsf{cid}, C, k_{sym}, \mathsf{pw}' \rangle$ to remember the symmetric key (see R.2). Note that Sim still gets the client's input pw' from $\mathcal{F}$, which we change in $\mathbf{G}_{14}$).

We further change how Sim reacts on a message $(\text{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, where C was computed by Sim itself for the retrieval of an honest client. Then, Sim does not retrieve $\mathsf{sk}_{\mathsf{enc}}$ to decrypt C. Instead, Sim retrieves the record $\langle \text{Ret}, \mathsf{ssid}, \mathsf{cid}, C, [k_{sym}], [\mathsf{pw}'] \rangle$ (RC.2 (a)) and proceeds as in $\mathbf{G}_6$ by executing the code of $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$. Additionally, before sending $(\text{RetRes}, \mathsf{ssid}, C')$ to S, Sim stores $\langle \mathsf{ssid}, C' \rangle$ (RC.2 (c)). Similarly to $\mathbf{G}_3$, the changes introduced here may lead to Sim decrypting C to a 0-string,

e.g., if $\mathcal{A}$ replays some C that was computed by SIM on behalf of some honest cid. Again, we let SIM output FAIL to S in that case (cf. RC.3 (b) and RC.7).

Note that the output behavior of SIM did not change. That is because SIM essentially behaves like $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ except that it does not encrypt and decrypt pw', k_{sym} but it stores these values and uses the stored values later. However, the distribution of C did change. Now, if the environment could distinguish $\mathbf{G}_7$ from $\mathbf{G}_6$, we can construct an adversary $\mathcal{B}_2$ against the IND-CCA security of PKE as follows:

Let $q_{\text{RET}} \in \mathbb{N}$ be the number of retrieval phases executed. First, $\mathcal{B}_2$ does not compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}})$ itself anymore but uses the pk^* provided by its challenger. We construct a sequence of games $\mathbf{G}_6^{(0)}, \ldots, \mathbf{G}_6^{(q_{\text{RET}})}$, where in $\mathbf{G}_6^{(i)}$ the first i ciphertexts that honest clients produce in retrieval are replaced by encryptions of $\perp$ and the remaining ciphertexts stay as in $\mathbf{G}_6$ (except that pk^* is used). If $\mathcal{Z}$ can distinguish $\mathbf{G}_7$ from $\mathbf{G}_6$ then there is an index $i^* \in [q_{\text{RET}}]$ such that $\mathcal{Z}$ distinguishes $\mathbf{G}_6^{(i^*)}$ and $\mathbf{G}_6^{(i^*-1)}$ with non-negligible advantage. $\mathcal{B}_2$ internally runs $\mathcal{Z}$ and simulates $\mathbf{G}_6^{(i^*-1)}$ except for the i^*-th retrieval. In this retrieval, $\mathcal{B}_2$ gives the messages $m_0 := (k_{sym}, \mathsf{pw}', \mathsf{cid}, \mathsf{ssid})$ and m_1 as a 0-string of the same length to its challenger. When the challenger responds with a ciphertext C^* then $\mathcal{B}_2$ uses C^* as the ciphertext in the message $(\text{RET}, \mathsf{ssid}, C^*, \mathsf{cid})$ that the honest client sends to S. Further, if $\mathcal{B}_2$ receives at some point a message $(\text{RET}, \mathsf{ssid}, C, \mathsf{cid})$ from a corrupted server to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, where there is no record $\langle \text{RET}, [\mathsf{ssid}'], [\mathsf{cid}'], C, [k_{sym}] \rangle$, i.e., the ciphertext towards the HSM is tampered with, then $\mathcal{B}_2$ gives C to the decryption oracle provided by the IND-CCA challenger. On the oracle's answer $(k_{sym}^*, \mathsf{pw}^*, \mathsf{cid}^*, \mathsf{ssid}^*)$ the reduction checks if $\mathsf{cid}^* = \mathsf{cid}$ and $\mathsf{ssid}^* = \mathsf{ssid}$. If these checks, fail the reduction gives output $(\mathsf{ssid}, \text{FAIL})$ to the server. Else, the reduction retrieves the record $\langle \text{FILE}, \mathsf{sid}, \mathsf{cid}, [\mathsf{pw}], [K], [\mathsf{ctr}] \rangle$. If $\mathsf{ctr} = 0$, then the reduction deletes the record and sends $(\text{DELREC}, \mathsf{cid})$ to S. Else, if $\mathsf{pw} = \mathsf{pw}^*$, the reduction sets $\mathsf{ctr} := 10$, computes $C' \xleftarrow{\$} \mathsf{SE.Enc}(k_{sym}^*, K)$, and sends (ssid, C') to S. Else, the reduction decrements ctr and sends $(\mathsf{ssid}, \text{FAIL})$ to S. Finally, $\mathcal{B}_2$ outputs whatever $\mathcal{Z}$ outputs.

Note that if C^* encrypts m_0, then the view of $\mathcal{Z}$ is distributed exactly as in $\mathbf{G}_6^{(i^*-1)}$, and if C^* encrypts m_1, then the view of $\mathcal{Z}$ is distributed exactly as in $\mathbf{G}_6^{(i^*)}$. We get

$$|\Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6]| \leq q_{\text{RET}} \mathbf{Adv}_{\mathsf{PKE},\mathcal{B}_2}^{\mathsf{IND\text{-}CCA}}(\lambda).$$

Game $\mathbf{G}_8$: Provide $\mathcal{F}$ with input of corrupted clients in retrieval. In this game we change SIM such that when it receives a message $(\text{RET}, \mathsf{ssid}, C, \mathsf{cid})$ from a corrupted cid the simulator now, in addition to decrypting $(k'_{sym}, \mathsf{pw}', \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$ and checking $\mathsf{cid} =$

$\mathsf{cid}', \mathsf{ssid} = \mathsf{ssid}'$, also gives input $(\text{RetC}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}')$ to $\mathcal{F}$. Similarly to $\mathbf{G}_5$, we use $\mathsf{pw}' = \bot$ if C is decrypted to a 0-string (RC.3 (a)). Note that the RetC interface of $\mathcal{F}$ is still a dummy interface, so this change does not alter the view of $\mathcal{Z}$, so

$$\Pr[\mathbf{G}_8] = \Pr[\mathbf{G}_7].$$

Game $\mathbf{G}_9$: Let $\mathcal{F}$ generate server output in retrieval. In this game, we change the RetC and RetS interfaces of $\mathcal{F}$ and we add the CompleteRetS interface to $\mathcal{F}$.

We change RetC and RetS such that they are as in $\mathcal{F}_{\mathsf{PPKR}}$ except that RetC still forwards the secret client input pw' to the simulator. Further, we change Sim such that it uses the interfaces to produce outputs for honest servers in the retrieval phase. That means, when Sim receives a message $(\text{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ from an honest or corrupt cid, it first proceeds as in $\mathbf{G}_8$, and before sending its output to cid, it sends the message $(\text{CompleteRetS}, \mathsf{ssid}, \mathsf{cid}, 1)$ to $\mathcal{F}$ (RC.4).

Additionally, Sim uses the CompleteRetS interface whenever the corrupt server honestly delivers a message from some honest cid to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$. Similarly to $\mathbf{G}_6$, this query gives its output back to Sim, but this query is again necessary to appropiately update the internal state of $\mathcal{F}$, in particular this interface resets the counter of cid to 10 if the retrieval is successful. Again, note that this requires the simulator to first give the input $(\text{RetS}, \mathsf{ssid}, \mathsf{cid}, \bot, \bot, \bot)$ to $\mathcal{F}$ (RC.5).

First, note that by definition, the interface RetC, resp. RetS, is called whenever an honest client, resp. honest server, starts a retrieval. Next, note that Sim also ensures that the RetC input is given to $\mathcal{F}$ when a corrupted client performs a retrieval. As Sim controls $\mathsf{sk}_{\mathsf{enc}}$, it is able to extract pw' by decrypting C and can give this as input to $\mathcal{F}$. Thus, $\mathcal{F}$ always retrieves a record $\langle \text{RetC}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}', \mathsf{pw}, K \rangle$ with $\mathsf{pw}' \neq \bot$ when Sim provides it with the CompleteRetS input. Consequently, if the provided password was correct, $\mathcal{F}$ outputs Succ to S and else it outputs Fail to S. Finally, note that $\mathcal{F}$ maintains the counter exactly as $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ did, i.e., the counter is set to 10 when a new File record is created, the counter is decremented when a password guess was wrong, and the counter is reset to 10 when a password guess was correct. Thus, from now on, Sim no longer keeps a counter in its File records. Overall, the output of an honest server did not change in this game and we get

$$\Pr[\mathbf{G}_9] = \Pr[\mathbf{G}_8].$$

Game $\mathbf{G}_{10}$: Switch symmetric ciphertext in retrieval. In this game, we change how Sim computes the ciphertext C' produced by the HSM during retrievals of honest clients. Instead of computing $C' \xleftarrow{\$} \mathsf{SE.Enc}(k_{sym}, K)$, where K is the retrieved key, the simulator now computes $C' \xleftarrow{\$}$

$\mathsf{SE.Enc}(k_{sym}, 0^\lambda)$ (RC.2 (b)). Note that k_{sym} is still chosen by SIM uniformly at random for every honest client that starts a retrieval.

Thus, if $\mathcal{Z}$ can distinguish $\mathbf{G}_9$ and $\mathbf{G}_{10}$, then we can construct an adversary $\mathcal{B}_3$ against the IND-CPA security of SE. We construct a sequence of games $\mathbf{G}_9^{(0)}, \ldots, \mathbf{G}_9^{(q_{\text{RET}})}$, where in $\mathbf{G}_9^{(i)}$ the first i ciphertexts C' are replaced by encryptions of $\perp$ as described above. If $\mathcal{Z}$ can distinguish $\mathbf{G}_9$ from $\mathbf{G}_{10}$, then there is an index i^* such that $\mathcal{Z}$ has non-negligible advantage in distinguishing $\mathbf{G}_9^{(i^*)}$ from $\mathbf{G}_9^{(i^*-1)}$. The reduction $\mathcal{B}_3$ internally runs $\mathcal{Z}$ and plays the role of $\mathcal{F}$ and SIM in $\mathbf{G}_9^{(i^*)}$ except for the i^*-th retrieval. There, $\mathcal{B}_3$ does not choose a symmetric key k_{sym} but sends the messages $m_0 := K$ and $m_1 := 0^\lambda$ to its challenger, where K is the backup key that the HSM would use to compute C'. When the challenger returns a ciphertext C^*, $\mathcal{B}_3$ outputs the message $(\text{RETRES}, \mathsf{ssid}, C^*)$.

Note that the view of $\mathcal{Z}$ in the case that C^* encrypts m_0 is distributed exactly as in $\mathbf{G}_9^{(i^*-1)}$ and in the case that C^* encrypts m_1 the view is distributed exactly as in $\mathbf{G}_9^{(i^*)}$. Therefore we get

$$|\Pr[\mathbf{G}_{10}] - \Pr[\mathbf{G}_9]| \leq q_{\text{RET}} \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{B}_3}(\lambda).$$

Game $\mathbf{G}_{11}$: Generate output for honest clients by $\mathcal{F}$. In this game, we change $\mathcal{F}$ and SIM such that SIM lets $\mathcal{F}$ produce the output for an honest client in the initialization and retrieval phase. To this end, SIM will make use of the interfaces COMPLETEINITC and COMPLETERETC that we add to $\mathcal{F}$ exactly as in $\mathcal{F}_{\mathsf{PPKR}}$. We change the following things:

- Whenever the simulator produces a message $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ on behalf of some honest client cid, the simulator no longer draws a random backup key K. Further, when it then receives a message $(\text{INITRES}, \mathsf{ssid}, \mathit{out})$ to cid, it uses the COMPLETEINITC interface with the bit b_C set accordingly (IO.2 and IO.1).
- Similarly, when SIM receives a message $(\text{RETRES}, \mathsf{ssid}, C')$ towards an honest client cid, the simulator now uses the COMPLETERETC interface with the bit b_C set depending on the value of out (RO.1 and RO.2).

As a consequence, SIM no longer needs to store the key K in the FILE records for honest initializations and the values pw and K in the INIT records.

Note that as of $\mathbf{G}_6$, $\mathcal{F}$ already chooses a uniformly random key K when it receives an INITC message, although that key is not used up to this game. Now, the COMPLETEINITC interface ensures that it is given as output to cid in the initialization phase. Similarly, in the retrieval phase of $\mathbf{G}_{10}$ the simulator retrieved its own stored backup key from the initialization phase to give it as output to cid in the retrieval phase. Now, $\mathcal{F}$, and not SIM, stores the backup key K in its FILE record, retrieves it when a retrieval is successful,

and gives it as output to cid or Sim when Sim sends a CompleteRetC message for a successful subsession.

Overall, the distribution of $\mathcal{Z}$'s view did not change and we get

$$\Pr[\mathbf{G}_{11}] = \Pr[\mathbf{G}_{10}].$$

Game $\mathbf{G}_{12}$: Retrieve maliciously initialized records from $\mathcal{F}$. In this game we change how Sim responds in retrievals by a corrupt party. When it receives a message $(\text{RetC}, \mathsf{ssid}, C^*, \mathsf{cid})$ from $\mathcal{A}$ on behalf of a corrupted cid, after checking $\mathsf{cid}' = \mathsf{cid}$ and $\mathsf{ssid}' = \mathsf{ssid}$, Sim now additionally sends $(\text{CompleteRetC}, \mathsf{ssid}, 1)$ to $\mathcal{F}$. If $\mathcal{F}$ answers with $(\text{RetRes}, \mathsf{ssid}, K')$, where $K \in \{\text{Fail}, \text{DelRec}\}$, then Sim forwards this to cid on behalf of S (RC.3 (c)). If $K' \notin \{\text{Fail}, \text{DelRec}\}$, then Sim needs to produce the ciphertext C'. However, the key K' received from $\mathcal{F}$ may be different from the key K that the corrupt cid chose in its last initialization, as $\mathcal{F}$ always chooses a random key when receiving the input InitC even for a corrupted cid. Therefore, if a record $\langle \text{File}, \mathsf{cid}, [K] \rangle$ exists, which implies that cid executed an initialization phase after being corrupted, then Sim encrypts K instead of K' to obtain C' (RC.3 (d) and RC.3 (e)). Note that this requires the simulator to delete any potentially existing File record of some cid whenever the corrupted S maliciously initializes for cid to ensure that it always returns the key of the most recent initialization (cf. IC.7).

Furthermore, we add the MaliciousRetrieve interface as in $\mathcal{F}_{\mathsf{PPKR}}$ to $\mathcal{F}$, which Sim now uses in retrievals by a corrupt S. When $\mathcal{A}$ instructs the corrupted server to send a message $(\text{RetC}, \mathsf{ssid}, C^*, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, where C^* was never sent by an honest client, which is again captured by checking if C^* decrypts to a 0-string (RC.7), Sim gives the input $(\text{MaliciousRetrieve}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}^*)$ to $\mathcal{F}$, where pw^* is the password it obtained from decrypting C^* (RC.8). If the response of $\mathcal{F}$ is $(\text{DelRec}, \mathsf{cid})$ or Fail, then Sim forwards the response to the server (RC.8 (a)). Else, $\mathcal{F}$ responds with K'. As before, we need to check whether the key returned by $\mathcal{F}$ is the correct one by checking whether a File record exists for cid (RC.8 (b)).

Note that the added interface does not output anything to the protocol parties but only to Sim. Further, note that $\mathcal{F}$ now behaves exactly as $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ did in $\mathbf{G}_{11}$ when a corrupted server interacts with $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$. More precisely, on a $(\text{MaliciousInit}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}^*, K^*)$ the functionality creates a record containing the password pw^* and the backup key K^* together with a counter, initialized to 10. Then on a $(\text{MaliciousRetrieve}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}^*)$ the functionality tries to retrieve this record, checks the counter, and compares the provided password with the stored password.

$$\Pr[\mathbf{G}_{12}] = \Pr[\mathbf{G}_{11}].$$

Game $\mathbf{G}_{13}$: Add unused attack interfaces. In this game, we add the interfaces LeakFile, corrupt, FullyCorrupt, and OfflineAttack to $\mathcal{F}$. Note that our simulator never uses these interfaces. That is because $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ does not leak any key files and cannot be corrupted. Thus, the protocol even realizes a version of $\mathcal{F}_{\mathsf{PPKR}}$ that does not allow offline attacks. We get

$$\Pr[\mathbf{G}_{13}] = \Pr[\mathbf{G}_{12}].$$

Game $\mathbf{G}_{14}$: Ideal world. We change the ideal functionality such that no more private input $\mathsf{pw}, \mathsf{pw}'$ is given to Sim. Note that Sim only used these inputs to determine if the password pw' used in a retrieval is correct. Instead it can now use the output *match* from $\mathcal{F}$ (RC.2 (b) and RC.5). Hence, we finally remove pw' from the Ret and File records. Also, we take away the ability of Sim to give output to parties. This is also not used anymore. Thus we get

$$\Pr[\mathbf{G}_{14}] = \Pr[\mathbf{G}_{13}].$$

After this change, we reached the ideal-world execution of the protocol encPw with the simulator $\text{Sim} = \text{Sim}_{\mathsf{encPw}}$ as described in Figures 5.3 and 5.4 and the functionality $\mathcal{F} = \mathcal{F}_{\mathsf{PPKR}}$ as described in Figures 4.7 and 4.8. In particular, we never had to change how Sim reacts on receiving a GetPK message when acting as $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$.

□

5.3 Lev-2 Protocol: Enhanced Encrypt-to-HSM

When the HSM leaks permanently stored protocol data such as account information of clients, also called "password files", one cannot hope to prevent an adversary from, e.g., offline-attacking a PPKR protocol. That is because the file must contain enough information for the HSM to decide whether a retrieval attempt is successful or not. Nonetheless, one can demand that an adversary still needs to guess a user's password and cannot read the password and/or the backup key immediately from the file. In other words, clients that choose very strong passwords should still have a certain level of security even if the HSM's files get leaked.

Clearly, the basic encrypt-to-HSM protocol from Section 5.2 falls short of this goal, and hence cannot reach Lev-2 security: The passwords and backup keys are stored in the clear and a compromise of the HSM's long-term state immediately gives them away. Further, the leakage includes all protocol-specific long-term state, i.e., the adversary also gets the HSM's decryption key $\mathsf{sk}_{\mathsf{enc}}$, which leaks all the password (attempts) from sessions happening *after* the compromise to the attacker.

To reach Lev-2 security, we strengthen encPw in two ways: First, we store the password only in hashed form at the HSM, with a user-specific salt to prevent

pre-computation attacks; and encrypt the backup key with another salted password hash. The salts are stored in the password file. Second, we let the HSM use ephemeral encryption keys for each session. Figure 5.5 shows our enhanced encrypt-to-HSM protocol encPw+. We can formally prove in Theorem 7 that these relatively simple (although in the case of the encryption keys significantly more expensive) measures are enough to restrict an adversary in Lev-2 back to offline guessing, after getting access to the long-term storage of the HSM.

We stress that our HSM leakage model at Lev-2 returns all permanently stored user files and protocol-specific *long-term* state to the adversary, but not the temporary values that occur during execution of the protocol.

Theorem 7. *Let* $\mathsf{SE} = (\mathsf{SE.KGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ *be an* IND-CPA*-secure symmetric encryption scheme,* $\mathsf{PKE} = (\mathsf{PKE.KGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an* IND-CCA*-secure public key encryption scheme,* $\mathsf{Sig} = (\mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Vrfy})$ *by an* sEUF-CMA*-secure signature scheme used for attestation, and* H *be modeled as a random oracle. Then, the protocol* encPw+ *from Figure 5.5 UC-realizes* $\mathcal{F}_{\mathsf{PPKR}}$ *at* Lev-2 *security (i.e., without the* FullyCorrupt *interface) in the* $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$*-hybrid model, malicious adaptive corruptions, and a client-authenticated channel between clients and the server. Concretely, we can construct adversaries* $\mathcal{B}_1, \ldots \mathcal{B}_4$ *such that for any efficient adversary against* encPw+ *(interacting with* $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$*), the simulator* $\textsc{Sim}_{\mathsf{encPw+}}$ *interacting with* $\mathcal{F}_{\mathsf{PPKR}}$ *produces a view such that for every efficient environment* $\mathcal{Z}$ *it holds that*

$$\begin{aligned}\mathbf{Adv}^{\mathcal{F}_{\mathsf{PPKR}}}_{\mathsf{encPw},\textsc{Sim}_{\mathsf{encPw+}},\mathcal{Z}}(\lambda) \leq\ & \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda) + q_{\textsc{Init}}\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_2}(\lambda) \\ & + q_{\textsc{Ret}}\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_3}(\lambda) + q_{\textsc{Ret}}\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SE},\mathcal{B}_4}(\lambda) + \frac{q_{\textsc{Init}}(2q_{\textsc{Init}}-1)}{2^\lambda} + \frac{q_H}{2^\lambda},\end{aligned}$$

where $q_{\textsc{Init}}$ *is the number of initializations,* $q_{\textsc{Ret}}$ *is the number of retrievals, and* q_H *is the number of* H *queries.*

Proof Sketch. Crucial for the proof of Theorem 6 and Theorem 7 is that the HSM is modeled as a hybrid-functionality. Therefore, the simulator will be able to extract malicious inputs using the HSM's secret encryption key. Further, honest clients can be simulated by invoking the CCA security of PKE as the client's passwords are not used elsewhere (except for determining the output). Finally, for Theorem 7, the simulator must deal with leaked files. To that end, the simulator can observe and program the random oracle H and use the OfflineAttack interface of $\mathcal{F}_{\mathsf{PPKR}}$. In case of a successful guess, the simulator learns K. Then, it can program $\mathsf{H}(s_2, \mathsf{pw}) \coloneqq c \oplus K$ to equivocate c after the fact, and similarly Sim can program $\mathsf{H}(s_1, \mathsf{pw}) \coloneqq h$.

The leaked client file does not contain values that would allow impersonation of the HSM (while full corruption, considered in Section 5.4, will allow HSM impersonation). □

Proof sketch. Because the formal proof of Theorem 7 shares most of its steps with the proof of Theorem 6, we focus on the changes to the proof that are

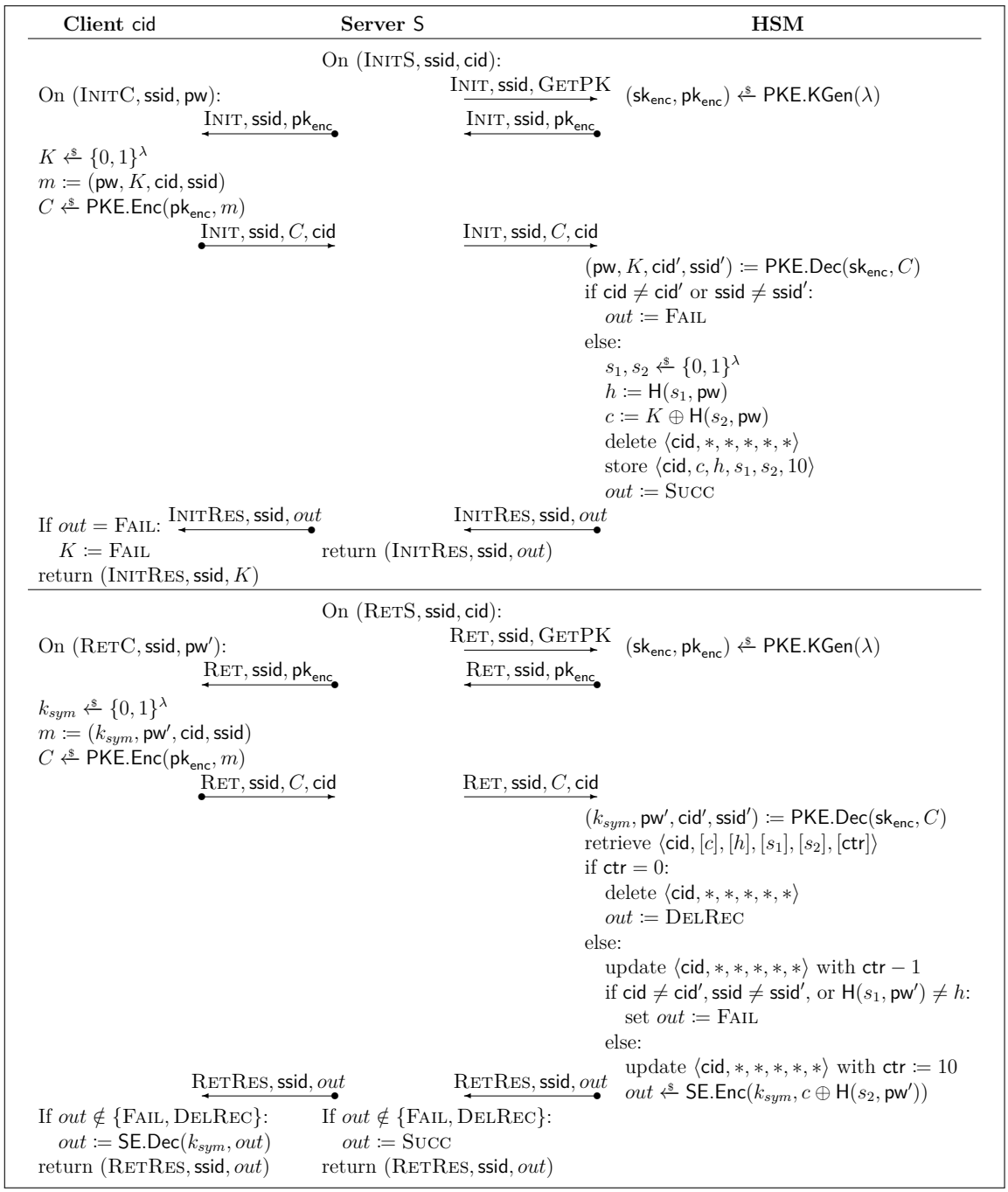

Figure 5.5: Protocol encPw+. $\xleftarrow{x}$ is the HSM-attested transmission of x. $\xrightarrow{y}$ is the client-to-server authenticated transmission of y. We implicitly assume that each message contains ssid and the random oracle H takes ssid as first input. cid and S output (InitRes, ssid, Fail) or (RetRes, ssid, Fail), whenever they receive an unexpected message (i.e. with mismatching ssid or cid) or when attestation verification fails. If any party receives the same type of message twice with the same ssid, it ignores the second one.

Functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$

On $(\textsc{Init}, \textsc{GetPK}, \mathsf{ssid})$ from S:
- $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$
- Store $\langle \mathsf{ssid}, \mathsf{sk_{enc}} \rangle$.
- Send $(\textsc{Init}, \mathsf{ssid}, \mathsf{pk_{enc}})$ to S.

On $(\textsc{Ret}, \textsc{GetPK}, \mathsf{ssid})$ from S:
- $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$
- Store $\langle \mathsf{ssid}, \mathsf{sk_{enc}} \rangle$.
- Send $(\textsc{Ret}, \mathsf{ssid}, \mathsf{pk_{enc}})$ to S.

On $(\textsc{Init}, \mathsf{ssid}, C, \mathsf{cid})$ from S:
- Retrieve $\langle \mathsf{ssid}, [\mathsf{sk_{enc}}] \rangle$
- $(\mathsf{pw}, K, \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk_{enc}}, C)$
- If $\mathsf{cid}' \neq \mathsf{cid}$ or $\mathsf{ssid}' \neq \mathsf{ssid}$ send $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Fail})$ to S. Else continue.
- $s_1, s_2 \xleftarrow{\$} \{0,1\}^\lambda$
- $h := \mathsf{H}(s_1, \mathsf{pw}), c := K \oplus \mathsf{H}(s_2, \mathsf{pw})$
- Delete $\langle \mathsf{cid}, *, *, *, *, * \rangle$ and store $\langle \mathsf{cid}, c, h, s_1, s_2, 10 \rangle$
- Delete record $\langle \mathsf{ssid}, \mathsf{sk_{enc}} \rangle$
- Send $(\textsc{InitRes}, \mathsf{ssid}, \textsc{Succ})$ to S

On $(\textsc{Ret}, \mathsf{ssid}, C, \mathsf{cid})$ from S:
- Retrieve $\langle \mathsf{ssid}, \mathsf{sk_{enc}} \rangle$.
- $(k_{sym}, \mathsf{pw}', \mathsf{cid}', \mathsf{ssid}') := \mathsf{PKE.Dec}(\mathsf{sk}, C)$
- Retrieve $\langle \mathsf{cid}, [c], [h], [s_1], [s_2], [\mathsf{ctr}] \rangle$
- If $\mathsf{ctr} = 0$, delete $\langle \mathsf{cid}, *, *, *, ** \rangle$ and send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{DelRec})$ to S. Else continue.
- If $\mathsf{cid}' \neq \mathsf{cid}$, $\mathsf{ssid}' \neq \mathsf{ssid}$, or $h \neq \mathsf{H}(s_1, \mathsf{pw}')$, update $\langle \mathsf{cid}, *, *, *, *, * \rangle$ with $\mathsf{ctr} - 1$ and send $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ to S. Else continue
- Update $\langle \mathsf{cid}, *, *, *, *, * \rangle$ with $\mathsf{ctr} := 10$.
- $C' \xleftarrow{\$} \mathsf{SE.Enc}(k_{sym}, c \oplus \mathsf{H}(s_2, \mathsf{pw}))$.
- Send $(\textsc{RetRes}, \mathsf{ssid}, C')$ to S

On $(\textsc{LeakFile})$ from $\mathcal{A}$:
- Set $L := \emptyset$
- For all $\langle [\mathsf{cid}], [c], [h], [s_1], [s_2], [\mathsf{ctr}] \rangle$ append $(\mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr})$ to L.
- Return L.

Figure 5.6: The ideal functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$.

required to also achieve security under leaked HSM files. We show the modified and additional interfaces that the simulator provides in Figure 5.8. The bulk of the proof works exactly as the proof of Theorem 6. But in contrast, we now have to simulate leakage of HSM files, because $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$ allows the adversary to call its LeakFile interface. To that end, the simulator must observe and program the random oracle H and use the OfflineAttack interface of $\mathcal{F}_{\mathsf{PPKR}}$. The simulator now also has to generate fresh encryption keys $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ for every initialization and retrieval and store them until they are used. Note that we can keep the rest of the simulation as in the proof of Theorem 6. That is because the leaked client file does not contain values that would allow impersonation of the HSM (as will be the case when we consider full corruption in Section 5.4).

Before the state of the HSM is leaked, an adversary has only a negligible chance to query $\mathsf{H}(s_2, \mathsf{pw})$ to the random oracle. That is because s_2 has high entropy. Therefore, the simulator can store a uniformly random value c as the encoding of the backup key. But once the HSM state is leaked, the adversary knows s_1, s_2 and h and thus, can guess passwords and verify its guess using h and s_1. If one of the guesses is correct, the adversary will be able to check if c is indeed an encoding of K by using s_2. The simulator can use the OfflineAttack interface of $\mathcal{F}_{\mathsf{PPKR}}$ to see if the adversary's guess was correct. In case of a successful guess, the simulator learns K. Then, it can program $\mathsf{H}(s_2, \mathsf{pw}) \coloneqq c \oplus K$ to equivocate c after the fact, and similarly Sim can program $\mathsf{H}(s_1, \mathsf{pw}) \coloneqq h$.

Proof. As in the proof of Theorem 6 we construct a sequence of hybrid games starting from the real world and ending in the ideal world. As the majority of game hops are identical to the previous proof, we only provide the games that are new. We write $\mathbf{G}_x$ for $x \in (i, i+1) \subset \mathbb{R}$ to denote that the game is added between game $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ in the sequence of games from the proof of Theorem 6.

Game $\mathbf{G}_{1.1}$: Abort on salt collision. In this game, the simulator aborts if it randomly samples a salt value that is already used for some other cid. More precisely, when Sim simulates $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$ and draws s_1 and s_2 uniformly at random during initialization, Sim checks if it already sampled either value in some previous initialization. If that is the case, Sim aborts the execution (see LF.3 (b)).

Let $q_{\text{Init}} \in \mathbb{N}$ be the number of initializations. We get

$$|\Pr[\mathbf{G}_{1.1}] - \Pr[\mathbf{G}_1]| \leq \frac{q_{\text{Init}}(2q_{\text{Init}} - 1)}{2^\lambda}.$$

Game $\mathbf{G}_{5.1}$: Add OfflineAttack and LeakFile interfaces to $\mathcal{F}$. In this game, we add the interfaces OfflineAttack and LeakFile to $\mathcal{F}$ exactly as they are in $\mathcal{F}_{\mathsf{PPKR}}$. Because Sim does currently not use them, we get

$$\Pr[\mathbf{G}_{5.1}] = \Pr[\mathbf{G}_5].$$

On $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

I.1 Wait for $(\text{INIT}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ from S to cid. $(\mathbf{G}_1)$

On $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

I.2 Compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and store $\langle \mathsf{ssid}, \mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}} \rangle$. Send $(\text{INIT}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ to cid on behalf of S. $(\mathbf{G}_1)$

On $(\text{INIT}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ from $\mathcal{A}$ to cid on behalf of S:

IPK.1 Compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, \perp)$ and store $\langle \text{INIT}, \mathsf{ssid}, \mathsf{cid}, C \rangle$ $(\mathbf{G}_3)$. Send $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ to S on behalf of cid. $(\mathbf{G}_1)$ // $\mathsf{pk}_{\mathsf{enc}}$ cannot be tampered with.

On $(\text{RETC}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

R.1 Wait for $(\text{RET}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ from S to cid. $(\mathbf{G}_1)$

On $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}', \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$:

R.2 Compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and store $\langle \mathsf{ssid}, \mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}} \rangle$. Send $(\text{RET}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ to cid on behalf of S. $(\mathbf{G}_1)$

On $(\text{RET}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ from $\mathcal{A}$ to cid on behalf of S:

RPK.1 Compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, \perp)$, choose $k_{sym} \xleftarrow{\$} \{0,1\}^\lambda$, and store $\langle \text{RET}, \mathsf{ssid}, \mathsf{cid}, C, k_{sym} \rangle$ $(\mathbf{G}_3)$. Send $(\text{RET}, \mathsf{ssid}, C, \mathsf{cid})$ to S on behalf of cid $(\mathbf{G}_1)$. // RETC came before $\mathsf{pk}_{\mathsf{enc}}$.

On $(\text{INIT}, \mathsf{ssid}, \text{GETPK})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of S:

G.1 Compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and store $\langle \mathsf{ssid}, \mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}} \rangle$. Send $(\text{INIT}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ to S on behalf of $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}+}$ $(\mathbf{G}_1)$. // attestated

On $(\text{RET}, \mathsf{ssid}, \text{GETPK})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of S:

G.2 Compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and store $\langle \mathsf{ssid}, \mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}} \rangle$. Send $(\text{RET}, \mathsf{ssid}, \mathsf{pk}_{\mathsf{enc}})$ to S on behalf of $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}+}$ $(\mathbf{G}_1)$. // attestated

On $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of cid:

IC.1 through IC.5 as in Figure 5.3.

IC.6 Try to retrieve all records $\langle \text{LEAKED}, \mathsf{cid}, [c], [h], [s_1], [s_2], *, * \rangle$ not marked old. If such records exist, then mark all of them as old. $(\mathbf{G}_{5.2})$ // After init, the file contains fresh values.

On $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}+}$ on behalf of S:

IC.6 and IC.7 as in Figure 5.3.

IC.8 Try to retrieve all records $\langle \text{LEAKED}, \mathsf{cid}, [c], [h], [s_1], [s_2], *, * \rangle$ not marked old. If such records exist, then mark all of them as old. $(\mathbf{G}_{5.2})$// After init, the file contains fresh values.

Figure 5.7: Differences from the simulator $\text{SIM}_{\mathsf{encPw}}$ (Figures 5.3 to 5.4) to the simulator $\text{SIM}_{\mathsf{encPw}+}$, part 1.

On a query (LeakFile) from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$:

LF.1 Give input (LeakFile) to $\mathcal{F}_{\mathsf{PPKR}}$. $\mathcal{F}_{\mathsf{PPKR}}$ returns L. ($\mathbf{G}_{5.2}$)

LF.2 $L \coloneqq \emptyset$ ($\mathbf{G}_{5.2}$)

LF.3 For each $(\mathsf{cid}, \mathsf{ctr}) \in L$:

a) Try to retrieve a record $\langle \text{LEAKED}, \mathsf{cid}, [c], [h], [s_1], [s_2], *, [i] \rangle$ not marked old. ($\mathbf{G}_{5.2}$)

b) If no such record exists, choose $c, h, s_1, s_2 \xleftarrow{\$} \{0,1\}^\lambda$ ($\mathbf{G}_{5.3}$ and $\mathbf{G}_{5.4}$). If s_1 or s_2 was used before as salt by Sim, then abort ($\mathbf{G}_{1.1}$). Else store $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, 1 \rangle$. Append $(\mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr})$ to L. ($\mathbf{G}_{5.2}$)

c) Else store a record $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, i+1 \rangle$, where i is the biggest number such that a record $\langle \text{LEAKED}, \mathsf{cid}, [c], [h], [s_1], [s_2], *, i \rangle$ exists. Append $(\mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr})$ to L. ($\mathbf{G}_{5.2}$)

LF.4 Send L to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$. ($\mathbf{G}_1$)

On a query $H(s^*, \mathsf{pw}^*)$:

H.1 If a record $\langle H, s^*, \mathsf{pw}^*, [y] \rangle$ exists, output y. ($\mathbf{G}_1$)

H.2 Try to retrieve the record $\langle \text{LEAKED}, [\mathsf{cid}], *, [h], s^*, *, *, [i] \rangle$. If such a record exists, give input (OfflineAttack, $\mathsf{cid}, \mathsf{pw}^*, i$) to $\mathcal{F}_{\mathsf{PPKR}}$. If $\mathcal{F}_{\mathsf{PPKR}}$ responds with $K \neq \text{Fail}$, record $\langle H, s^*, \mathsf{pw}^*, h \rangle$ and output h. If $K = \text{Fail}$, choose $y \xleftarrow{\$} \{0,1\}^\lambda$, record $\langle H, s^*, \mathsf{pw}^*, y \rangle$ and output y. ($\mathbf{G}_{5.3}$)

H.3 If no such LEAKED record exists, try to retrieve the record $\langle \text{LEAKED}, [\mathsf{cid}], [c], *, *, s^*, *, [i] \rangle$. If such a record exists, give input (OfflineAttack, $\mathsf{cid}, \mathsf{pw}^*, i$) to $\mathcal{F}_{\mathsf{PPKR}}$. If $\mathcal{F}_{\mathsf{PPKR}}$ responds with $K \neq \text{Fail}$, record $\langle H, s^*, \mathsf{pw}^*, c \oplus K \rangle$ and output $c \oplus K$. If $K = \text{Fail}$, choose $y \xleftarrow{\$} \{0,1\}^\lambda$, record $\langle H, s^*, \mathsf{pw}^*, y \rangle$ and output y. ($\mathbf{G}_{5.4}$)

H.4 If both such LEAKED do not exist ($\mathbf{G}_{5.4}$), choose $y \xleftarrow{\$} \{0,1\}^\lambda$, record $\langle H, s^*, \mathsf{pw}^*, y \rangle$ and output y ($\mathbf{G}_1$).

Figure 5.8: Differences from the simulator $\text{Sim}_{\mathsf{encPw}}$ (Figures 5.3 to 5.4) to the simulator $\text{Sim}_{\mathsf{encPw+}}$, part 2.

Game $\mathbf{G}_{5.2}$: Use LeakFile interface of $\mathcal{F}$. In this game, we change how Sim responds to a (LeakFile) message from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$. Sim gives input (LeakFile) to $\mathcal{F}$ and receives a list L from $\mathcal{F}$ (see LF.1). The simulator initializes $L := \emptyset$. Now, for all $(\mathsf{cid}, \mathsf{ctr}) \in L$ the simulator tries to retrieve a record $\langle \text{LEAKED}, \mathsf{cid}, [c], [h], [s_1], [s_2], *, * \rangle$ that is not marked old.

If no such record exists then Sim chooses $s_1, s_2 := \{0,1\}^\lambda$ uniformly at random. Then, Sim computes $h := \mathsf{H}(s_1, \mathsf{pw})$ and $c := K \oplus \mathsf{H}(s_2, \mathsf{pw})$ and creates a record $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, 1 \rangle$. Sim appends $(\mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr})$ to L.

Otherwise, Sim adds $(\mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr})$ to L and records $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, i+1 \rangle$, where i is the biggest number such that a record $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, *, i \rangle$ exists.

Sim also does an additional step when an initialization was completed, i.e., in IC.6 or IC.8, depending on whether the server is corrupted or not. Sim goes through all so far leaked records and marks them as old. That means that they still can be offline attacked, but the next time when a record is leaked, Sim will simulate the creation of a new file. In particular, Sim keeps LEAKED records of already overwritten initializations. This will be important for programming H later. However, the records that are added to L are always the "current" records, i.e., records that correspond to File records of $\mathcal{F}$.

Also, note that Sim only appends leaked data for cid if $\mathsf{cid} \in L$. However, in the previous games, we ensured that $\mathcal{F}$ creates a File record whenever $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$ would have done so. More precisely, when Sim receives a message $(\text{Init}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of a corrupted cid and all the checks pass then Sim gives input $(\text{CompleteInitC}, \mathsf{ssid}, \mathsf{cid}, 1)$ to $\mathcal{F}$ so $\mathcal{F}$ creates a File record. Similarly, when Sim receives a message $(\text{Init}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw+}}$ on behalf of S and all the checks pass then Sim sends either a MaliciousInit message or $(\text{CompleteInitS}, \mathsf{ssid}, \mathsf{cid}, 1)$ to $\mathcal{F}$ so $\mathcal{F}$ creates a File record.

Overall, the view of $\mathcal{Z}$ did not change.

$$\Pr[\mathbf{G}_{5.2}] = \Pr[\mathbf{G}_{5.1}].$$

Game $\mathbf{G}_{5.3}$: Simulate the password hash h. In this game, we let Sim simulate $h = \mathsf{H}(s_1, \mathsf{pw})$ without using pw and we let Sim program the random oracle accordingly.

First, when Sim creates a record $\langle \text{LEAKED}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, i \rangle$ it no longer computes h like in the real protocol but now chooses it uniformly at random, i.e., $h \xleftarrow{\$} \{0,1\}^\lambda$ (see LF.3 (b))

Second, Sim observes the random oracle queries. If there is a query $\mathsf{H}(s_1^*, \mathsf{pw}^*)$ such that a record $\langle \text{LEAKED}, [\mathsf{cid}], *, [h], s_1^*, *, *, [i] \rangle$ exists, then

Sim gives input $(\text{OfflineAttack}, \mathsf{cid}, \mathsf{pw}^*, i)$ to $\mathcal{F}$. If $\mathcal{F}$ answers with Fail, then Sim responds with a uniformly random value. If $\mathcal{F}$ answers with $K \neq \text{Fail}$, then Sim programs $\mathsf{H}(s_1^*, \mathsf{pw}^*)$ to h (see H.2). This ensures that if $\mathcal{A}$ guesses the password of some client whose file was leaked previously, then the output of the random oracle query is consistent with the values that Sim output in the leaked file.

Since H is modeled as a random oracle, choosing h as a uniformly random value does not modify the distribution of h. However, observe that $\mathcal{Z}$ could distinguish this game from the previous one if $\mathcal{A}$ guesses the salt s_1^* that some cid uses together with the password pw^* before leaking the file of cid. If that happens, Sim would output a random value h^* on the oracle query (s_1^*, pw^*). Then, upon $\mathcal{A}$ leaking the file of cid, the simulator would not output h^* in the leaked file and instead choose a random value h since it cannot know that cid chose the password pw^*. As s_1 is chosen uniformly at random from $\{0,1\}^\lambda$, the probability of this happening can be bounded by $q_H 2^{-\lambda}$, where $q_H \in \mathbb{N}$ is the number of queries to H. Note that this even holds if other clients use the same pw^* because due to $\mathbf{G}_{1.1}$ they will use a different salt s_1' and $\mathsf{H}(s_1', \mathsf{pw}^*)$ is independent of $\mathsf{H}(s_1^*, \mathsf{pw}^*)$. Thus, we have

$$|\Pr[\mathbf{G}_{5.3}] - \Pr[\mathbf{G}_{5.2}]| \leq \frac{q_{\text{Init}} q_H}{2^\lambda}.$$

Game $\mathbf{G}_{5.4}$: Simulate the password hash c. In this game, we let Sim simulate the leaked records without using K and pw and we let Sim program the random oracle accordingly.

First, when Sim creates a record $\langle \text{leaked}, \mathsf{cid}, c, h, s_1, s_2, \mathsf{ctr}, i \rangle$ it no longer computes c like in the real protocol but now chooses it uniformly at random, i.e., $c \coloneqq \{0,1\}^\lambda$ (see LF.3 (b)).

Second, Sim observes the random oracle queries. If there is a query $\mathsf{H}(s_2^*, \mathsf{pw}^*)$ such that a record $\langle \text{leaked}, [\mathsf{cid}], [c], *, *, s_2^*, *, [i] \rangle$ exists, then Sim gives input $(\text{OfflineAttack}, \mathsf{cid}, \mathsf{pw}^*, i)$ to $\mathcal{F}$. If $\mathcal{F}$ answers with Fail, then Sim responds with a uniformly random value. If $\mathcal{F}$ answers with $K \neq \text{Fail}$, then, Sim programs $\mathsf{H}(s_2^*, \mathsf{pw}^*) \coloneqq c \oplus K$. Note that for honest cid, Sim uses the key K it chose on behalf of cid in the initialization instead of the one returned by $\mathcal{F}$. Once we let $\mathcal{F}$ generate the output of honest clients in $\mathbf{G}_{11}$, we change this to use the key returned from $\mathcal{F}$. This again ensures that the leaked files are consistent with the random oracle outputs (H.3).

Again, it holds that the distribution of c does not change by choosing it as uniformly random value since H is a random oracle. However, similarly to $\mathbf{G}_{5.3}$, the environment could distinguish this game from the previous one if $\mathcal{A}$ guesses the salt s_2^* of some cid^*. Following the same arguments as in $\mathbf{G}_{5.3}$, we have

$$|\Pr[\mathbf{G}_{5.4}] - \Pr[\mathbf{G}_{5.3}]| \leq \frac{q_{\text{Init}} q_H}{2^\lambda}.$$

Changed Reductions Also, we have to change the reductions on IND-CCA security in $\mathbf{G}_3$ and $\mathbf{G}_7$ because of the difference in computing $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}})$:

Game $\mathbf{G}_3$: Let $q_{\text{INIT}} \in \mathbb{N}$ be the number of initializations. We construct a sequence of games $\mathbf{G}_1^{(0)}, \ldots, \mathbf{G}_1^{(q_{\text{INIT}})}$, where in $\mathbf{G}_1^{(i)}$ the first i ciphertexts are computed as encryptions of 0-strings with appropriate length and the remaining ciphertexts are encrypted as in $\mathbf{G}_1$. We have $\mathbf{G}_1 = \mathbf{G}_1^{(0)}$ and $\mathbf{G}_2 = \mathbf{G}_1^{(q_{\text{INIT}})}$. Because $\mathcal{Z}$ can distinguish $\mathbf{G}_1$ from $\mathbf{G}_2$, there must be an index $i^* \in [q_{\text{INIT}}]$ such that $\mathcal{Z}$ has a non-negligible advantage in distinguishing $\mathbf{G}_1^{(i^*)}$ and $\mathbf{G}_1^{(i^*-1)}$. Now, in the i^*-th initialization the reduction $\mathcal{B}_2$ does not choose a new key-pair $(\mathsf{sk}, \mathsf{pk})$ when it receives a GETPK or INITS message, but uses the public key that it is provided by the challenger. When it simulates cid in the i^*-th simulation, it gives $m_0 := (k_{sym}, \mathsf{pw}', \mathsf{cid}, \mathsf{ssid})$ and m_1 as a 0-string of the same length to the challenger and uses the returned C^* as the ciphertext for the initialization. Further, if $\mathcal{B}_2$ receives a message $(\text{INIT}, \mathsf{ssid}, C, \mathsf{cid})$ from $\mathcal{A}$ to S or to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ such that $C \neq C^*$, then $\mathcal{B}_2$ uses its decryption oracle to decrypt C^* and proceeds as in $\mathbf{G}_1$.

Now, if C^* encrypts m_0 the game is distributed as in $\mathbf{G}_1^{(i^*-1)}$ and if it encrypts m_1 then the game is distributed as in $\mathbf{G}_1^{(i^*)}$.

Game $\mathbf{G}_7$: Let $q_{\text{RET}} \in \mathbb{N}$ be the number of retrieval phases executed. We construct a sequence of games $\mathbf{G}_5^{(0)}, \ldots, \mathbf{G}_5^{(q_{\text{RET}})}$, where in $\mathbf{G}_5^{(i)}$ the first i ciphertexts that honest clients produce in retrieval are replaced by encryptions of 0-strings of appropriate length and the remaining ciphertexts stay as in $\mathbf{G}_5$. If $\mathcal{Z}$ can distinguish $\mathbf{G}_6$ from $\mathbf{G}_5$, then there is an index $i^* \in [q_{\text{RET}}]$ such that $\mathcal{Z}$ distinguishes $\mathbf{G}_5^{(i^*)}$ and $\mathbf{G}_5^{(i^*-1)}$ with non-negligible advantage. $\mathcal{B}_3$ internally runs $\mathcal{Z}$ and simulates $\mathbf{G}_5^{(i^*-1)}$ except for the i^*-th retrieval. In this retrieval, $\mathcal{B}_3$ does not choose a fresh pair $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ when it receives a GETPK message, but uses the public key $\mathsf{pk}_{\mathsf{Enc}}^*$ provided by the challenger. $\mathcal{B}_3$ gives the messages $m_0 := (\mathsf{pw}, K, \mathsf{cid}, \mathsf{ssid})$ and m_1 as a 0-string of the same length to its challenger. When the challenger responds with a ciphertext C^*, then $\mathcal{B}_3$ uses C^* as the ciphertext in the message $(\text{RET}, \mathsf{ssid}, C^*, \mathsf{cid})$ that the honest client sends to S. Further, if $\mathcal{B}_3$ receives at some point a message $(\text{RET}, \mathsf{ssid}, C, \mathsf{cid})$ to S or $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$, where $C \neq C^*$, then $\mathcal{B}_3$ gives C to the decryption oracle provided by the IND-CCA challenger and proceeds as in $\mathbf{G}_5$. Finally, $\mathcal{B}_3$ outputs whatever $\mathcal{Z}$ outputs.

Note that if C^* encrypts m_0 then the view of $\mathcal{Z}$ is distributed exactly as in $\mathbf{G}_5^{(i^*-1)}$ and if C^* encrypts m_1 then the view of $\mathcal{Z}$ is distributed exactly as in game $\mathbf{G}_5^{(i^*)}$. □

Missing Lev-3 Security of encPw+. Cleary, our enhanced encrypt-to-HSM protocol cannot satisfy Lev-3 security: if the adversary gets the HSM's attestation key, it has full control over the public keys for the encryption scheme under which the client encrypts the user's password. Thus, for all users that still use the PPKR service after the full compromise happened, the adversary immediately learns their plaintext passwords from the request – and consequently can retrieve their keys too.

To achieve Lev-3 security, we need to securely communicate some password-dependent data to a possibly entirely malicious party for authentication, and cannot rely on any server-held secret for the password's protection. This requires more advanced cryptographic techniques, for which we turn to an oblivious pseudorandom function in our third construction.

5.4 Lev-3 Protocol: OPRF-based PPKR

In this section, we build PPKR that provides protection against full server and HSM corruption. That is, even when all keys and files are leaked, the best an adversary can do is an offline attack against each user's password. Our goal is to propose a simpler protocol than WBP that reaches such Lev-3 security.

Oblivious Pseudorandom Function. At the core of our construction is an oblivious pseudorandom function (OPRF). Such a function allows to deterministically compute $y = \mathsf{PRF}(k, x)$ through an interactive protocol between an evaluator and requester. The evaluator knows the PRF key k, but learns nothing about the input x or output y it computes. We will use such an OPRF to deterministically compute password-dependent key material, which allows the client to authenticate towards the HSM, and also derive the encryption key under which K gets wrapped.

High-Level Idea of OPRF-PPKR. For initialization, the client executes the OPRF with the HSM (through the server) to obtain a value $\rho = \mathsf{PRF}(\mathsf{k}_{\mathsf{OPRF}}, (\mathsf{pw}, \mathsf{cid}))$, depending on the user's password and cid. The HSM chooses a client-specific key $\mathsf{k}_{\mathsf{OPRF}}$. The derived ρ then serves as the key for an authenticated encryption (AE) scheme, under which the client's randomly chosen key K gets encrypted. The client also generates a signature key pair $(\mathsf{sk}_C, \mathsf{pk}_C)$ for future re-authentication. It encrypts both sk_C and K under ρ, obtaining a ciphertext c. The HSM gets c and pk_C and stores them with $\mathsf{cid}, \mathsf{k}_{\mathsf{OPRF}}$, and a counter value ctr, initially set to 10, forming the password file for cid.

For retrieval, the client receives c from the HSM and runs the OPRF again. If the client uses the same password as in initialization, it obtains the same ρ as earlier. The client then uses ρ to decrypt c, obtaining (K, sk_C). Now the client can prove knowledge of the right password towards the HSM, by using the sk_C to sign the transcript of previous messages exchanged with the HSM. The HSM decreases the counter ctr at the beginning of every retrieval, and if it receives such a signature that verifies under the pk_C in the stored user file, it considers the client as correctly authenticated and resets the ctr value to 10 again.

Interestingly, we still need a fresh encryption key pair for communication towards the HSM – but only for the initialization. Therein, the client's values (pk_C, c) and the ssid get encrypted under the freshly chosen key. The purpose of this encryption is to prevent a malicious server from combining the ciphertext c of

an honest user with a different public key pk_C. Such a mixing attack would allow the server to plant the honest c with some authentication information where it could easily circumvent the strict limit of failed retrievals imposed by the HSM. Thus, the purpose of the public-key encryption here is to bind the honest user's information together.

Comparison to WBP. Our protocol shares a lot of similarities with the WBP, but simplifies the design enabling both a more efficient and more secure variant. We discuss the main differences to our protocol here.

Recall that the overall idea of the WBP is to let the client and the HSM execute an asymmetric PAKE protocol to exchange fresh symmetric key $(\mathsf{K}_C^{\mathsf{MAC}}, \mathsf{K}_S^{\mathsf{MAC}}, \mathsf{shk})$ from the password of the client, and the password file stored by the HSM. $\mathsf{K}_C^{\mathsf{MAC}}$ is subsequently used to prove the correctness of the client's password to the HSM via a MAC, to reset the file counter. The client's key K is encrypted under the key K^{export}, which is derived with an OPRF, and stored by the HSM in the ciphertext e. The protocol is instantiated with OPAQUE [JKX18], which in turn crucially relies on an OPRF to produce the static export key.

In our protocol, we start from an OPRF instead of aPAKE, and implement the proof of password knowledge separately through digital signatures. This yields a simpler protocol layout and cleaner security proof. In terms of similarity, the AE ciphertext stored in the client's password files corresponds to the OPAQUE password files, and the OPRF is used to deterministically derive a key to decrypt this file and perform the re-authentication. How this authentication is done differs though: we save one message by using signatures instead of authenticated key exchange. Note that this exploits that we are not aiming at the key exchange, which was the goal of OPAQUE – and thus is more than what is needed for PPKR.

Apart from improving efficiency, our protocol provides better security then WBP. While the following attacks can all be considered minor, they are still in conflict with the desired security properties – and easily preventable as shown by our protocol. The security improvements are as follows:

1. In the WBP, a corrupt server can prevent old password files of honest clients from being overwritten upon a client re-initializing with a new password. The corrupt server could then still use up all remaining password guesses against old password files, to retrieve previous keys of the client (cf. Section 4.2.6). We prevent this by having the HSM attest the client identity cid for each session ssid, and letting the client abort when it receives a mismatching cid. This enforces an agreement between an honest client and honest HSM when the server is corrupt (needed for Lev-1 security).

2. We fix a "rerouting" attack on the WBP which allows a corrupt server to reroute honest retrieval attempts to wrong password files. This results in honest Alice retrieving Bob's key if both use the same password. We therefore invoke the OPRF not only on the user's password but also append

the unique client identity cid to the input. This simple measure ensures that users derive unique wrapping keys ρ, which protects against this attack (again needed for Lev-1 security).

3. The above two measures additionally prevent two attacks on the WBP that a fully corrupt server in the Lev-3 setting[2] can mount: (I) checking whether two honest clients use the same password, (II) resetting the counter in an honest client's password file if another client having the same password runs a retrieval. Both attacks work by routing a retrieval attempt of Alice to Bob's password file, which in the WBP goes unnoticed by Alice, and can succeed because the passwords of honest clients are not enforced to be unique. Both is prevented through measures (1) and (2).

4. Considering the same HSM leakage model that we use (everything except the HSM's attestation key gets leaked on Lev-2), in the WBP, a malicious server can run offline password guessing attack not only against all leaked files but also against files that are created *after* the compromise. This is because the WBP relies on a long-term encryption key at the HSM for initialization. After a Lev-2 compromise, the malicious server can decrypt an honest user's initialization request, and re-encrypt the user's wrapped key K together with a maliciously chosen (AKE) public key. This allows the malicious server to get unlimited password guesses against the user's real password-wrapped key, as it can correctly complete key confirmation towards the HSM, even when performing the retrieval with the wrong passwords. Our protocol prevents that attack by using fresh encryption keys in initialization, achieving the necessary security for Lev-2.

Concrete OPRF for Efficiency. While our protocol can be securely realized with any 2-round OPRF, we build OPRF-PPKR from the 2HashDH OPRF [JKKX16] in a non-black-box way. This was mainly done for efficiency reasons. A generic approach relying on an ideal OPRF functionality would not allow binding the OPRF in- and outputs directly to other protocol values, nor do this efficiently in an HSM-attested way. However, even though our protocol is non-generic, our proof actually provides some modularity: we use the 2HashDH simulator as a step in our proof, and from then on rely on the abstracted security properties of a UC-secure OPRF. Nevertheless, our proof additionally relies on the internals of the 2HashDH simulator at several points, making the switch to another OPRF non-trivial.

Before stating our theorem, we introduce the security notion of *equivocability* [JKX18] for authenticated encryption that we require in our construction. Equivocability means that a simulator is able to produce ciphertexts of the scheme

[2] Although we did not formally analyze the WBP under full corruptions, t is easy to verify that a fully corrupt server in the WBP can use the same OPRF key for Alice and Bob and perform these attacks.

without committing to the plaintext. If the ciphertext must be decrypted then the simulator can output an appropriate key to open the ciphertext to a message of choice.

Definition 35. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against the *equivocability* (EQV) of an AE scheme $\mathsf{AE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

$$\mathbf{Adv}^{\mathsf{EQV}}_{\mathsf{SE},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}_1(c,k) = 1 \;\middle|\; \begin{array}{c} m \xleftarrow{\$} \mathcal{A}_0 \\ k \xleftarrow{\$} \mathsf{KGen}(1^\lambda) \\ c \xleftarrow{\$} \mathsf{Enc}(k,m) \end{array} \right] - \Pr\left[\mathcal{A}_1(c,k) = 1 \;\middle|\; \begin{array}{c} m \xleftarrow{\$} \mathcal{A}_0 \\ c \xleftarrow{\$} \textsc{Sim}^{(1)}_{\mathsf{EQV}}(|m|) \\ k \xleftarrow{\$} \textsc{Sim}^{(2)}_{\mathsf{EQV}}(c,m) \end{array} \right] \right|.$$

We say AE is EQV-secure if for any efficient $\mathcal{A}$, there exists an efficient simulator $\textsc{Sim}_{\mathsf{EQV}} = (\textsc{Sim}^{(1)}_{\mathsf{EQV}}, \textsc{Sim}^{(2)}_{\mathsf{EQV}})$ such that $\mathbf{Adv}^{\mathsf{EQV}}_{\mathsf{AE},\mathcal{A}}(\lambda)$ is negligible in λ.

We refer the reader to [JKX18] for an instantiation of an authenticated encryption scheme that is equivocable and random-key robust.

Theorem 8. *Let* $\mathsf{AE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an* IND-CPA-*,* INT-CTXT-*,* RKR-*, and* EQV*-secure authenticated encryption scheme,* $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an* IND-CCA*-secure public key encryption scheme,* $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *an* sEUF-CMA*-secure signature scheme, and* H_1 *and* H_2 *be modeled as a random oracles. Then, the protocol* OPRF-PPKR *from Figure 5.9 UC-realizes* $\mathcal{F}_{\mathsf{PPKR}}$ *at* Lev-3 *security in the* $\mathcal{F}^{\mathsf{OPRF\text{-}PPKR}}_{\mathsf{HSM}}$*-hybrid model, malicious adaptive corruptions, and a client-authenticated channel between clients and the server. Concretely, we can construct adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_5$*, and environment* $\mathcal{Z}_1$ *such that for any efficient adversary against* OPRF-PPKR *(interacting with* $\mathcal{F}^{\mathsf{OPRF\text{-}PPKR}}_{\mathsf{HSM}}$*), the simulator* $\textsc{Sim}_{\mathsf{OPRF\text{-}PPKR}}$ *interacting with* $\mathcal{F}_{\mathsf{PPKR}}$ *produces a view such that for every efficient environment* $\mathcal{Z}$ *it holds that*

$$\begin{aligned} \mathbf{Adv}^{\mathcal{F}_{\mathsf{PPKR}}}_{\mathsf{OPRF\text{-}PPKR},\textsc{Sim}_{\mathsf{OPRF\text{-}PPKR}},\mathcal{Z}}(\lambda) \leq & \mathbf{Adv}^{\mathcal{F}_{\mathsf{OPRF}}}_{\mathsf{2HDH},\textsc{Sim}_{\mathsf{2HDH}},\mathcal{Z}_1}(\lambda) + \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda) \\ & + q_E^2 \mathbf{Adv}^{\mathsf{RKR}}_{\mathsf{AE},\mathcal{B}_2}(\lambda) + q_{\textsc{Init}} \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AE},\mathcal{B}_3}(\lambda) + \mathbf{Adv}^{\mathsf{EQV}}_{\mathsf{AE},\mathcal{B}_4}(\lambda) \\ & + q_{\textsc{Init}} \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_5}(\lambda) + q_{\textsc{Init}} \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}_6}(\lambda) \end{aligned}$$

where $q_{\textsc{Init}}$ *is the number of initializations,* $q_{\textsc{Ret}}$ *is the number of retrievals.*

Proof Sketch. The proof heavily relies on the security of the 2HashDH OPRF. We reuse the simulator $\textsc{Sim}_{\mathsf{2HDH}}$ from Section 4.4.1, which demonstrates that 2HashDH UC-realizes $\mathcal{F}_{\mathsf{OPRF}}$, in a non-black-box way throughout the proof. Whenever we have to simulate a message a, b, a', or b' for some honest party, we "outsource" the simulation to $\textsc{Sim}_{\mathsf{2HDH}}$. This also means that we let $\textsc{Sim}_{\mathsf{2HDH}}$ choose the key $\mathsf{k}_{\mathsf{OPRF}}$, which we then need to obtain from $\textsc{Sim}_{\mathsf{2HDH}}$ whenever there is a LeakFile query from $\mathcal{A}$. Furthermore, from the random oracle queries to H_2 by $\mathcal{A}$, $\textsc{Sim}_{\mathsf{2HDH}}$ is able to extract the password chosen by a corrupt party in an initialization, which allows us to install a corresponding file in $\mathcal{F}_{\mathsf{PPKR}}$.

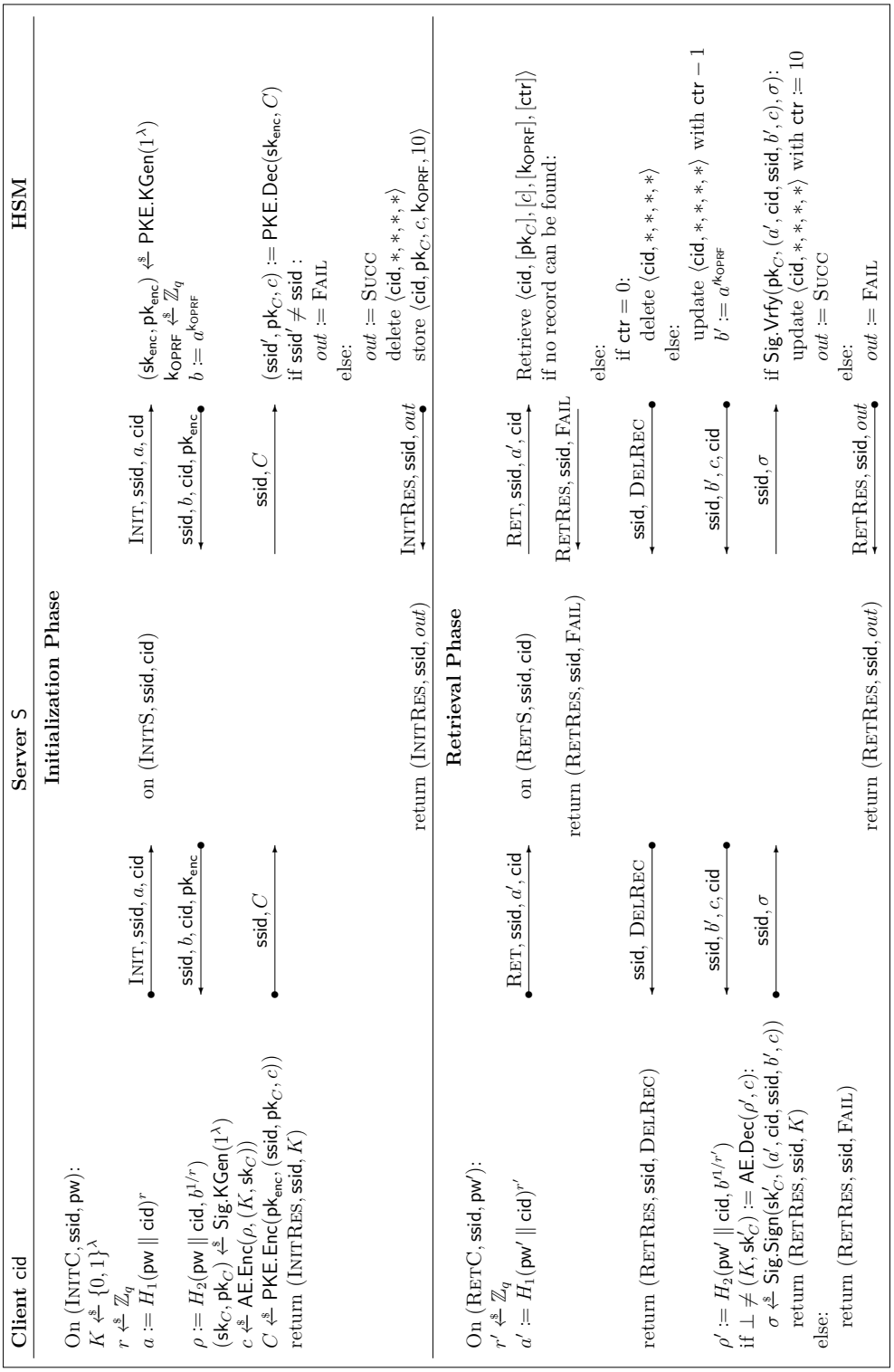

Figure 5.9: Protocol OPRF-PPKR. $\overset{x}{\underset{y}{\longleftarrow\!\bullet}}$ indicates that the message x is signed by the HSM; and $\bullet\!\longrightarrow$ denotes the client-to-server authenticated transmission of y. cid outputs (InitRes, ssid, Fail) or (RetRes, ssid, Fail), whenever it receives an unexpected message (i.e. with mismatching ssid or cid) or when signature verification fails for the received signed message. If any party receives the same type of message twice with the same ssid it ignores the second one.

Functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$

The functionality initially computes $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Sig.KGen}(1^\lambda)$

On (GetPK, ssid) from anyone:
- Output pk.

On (Init, ssid, a, cid) from S:
- $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) := \mathsf{PKE.KGen}(1^\lambda)$
- $\mathsf{k}_{\mathsf{OPRF}} \xleftarrow{\$} \mathbb{Z}_q$, $b := a^{\mathsf{k}_{\mathsf{OPRF}}}$
- send $(\mathsf{ssid}, b, \mathsf{cid}, \mathsf{pk}_{\mathsf{enc}})$ to S

On (ssid, C) from S:
- $(\mathsf{ssid}', \mathsf{pk}_C, c) := \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$
- if $\mathsf{ssid}' \neq \mathsf{ssid}$: $out :=$ Fail
  else:
    store $\langle \mathsf{cid}, \mathsf{pk}_C, c, \mathsf{k}_{\mathsf{OPRF}}, \mathsf{ctr} := 10 \rangle$, $out :=$ Succ
- send (InitRes, ssid, out) to S

On (Ret, ssid, a', cid) from S:
- Retrieve $\langle \mathsf{cid}, [\mathsf{pk}_C], [c], [\mathsf{k}_{\mathsf{OPRF}}], [\mathsf{ctr}] \rangle$
- if no record can be found: send (RetRes, ssid, Fail) to S
  else:
    if $\mathsf{ctr} = 0$: delete $\langle \mathsf{cid}, \mathsf{pk}_C, c, \mathsf{k}_{\mathsf{OPRF}}, \mathsf{ctr} \rangle$
    else:
      set ctr in the record to $\mathsf{ctr} - 1$, $b' := a'^{\mathsf{k}_{\mathsf{OPRF}}}$
      send $(\mathsf{ssid}, b', c, \mathsf{cid})$ to S

On (ssid, σ) from S:
- if $\mathsf{Sig.Vrfy}(\mathsf{pk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c), \sigma)$: set counter in the record to $\mathsf{ctr} := 10$. Set $out :=$ Succ
  else:
    $out :=$ Fail
- send (RetRes, ssid, out) to S

On (LeakFile, sid) from $\mathcal{A}$:
- Set $L := \emptyset$
- For every record $\langle [\mathsf{cid}], [\mathsf{pk}_C], [c], [\mathsf{k}_{\mathsf{OPRF}}], [\mathsf{ctr}] \rangle$ append $(\mathsf{cid}, \mathsf{pk}_C, c, \mathsf{k}_{\mathsf{OPRF}}, \mathsf{ctr})$ to L.
- Output L to $\mathcal{A}$.

On (FullyCorrupt, sid) from $\mathcal{A}$:
- Run (LeakFile, sid) to obtain L.
- Output L and sk.

Figure 5.10: The ideal functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$.

A challenge that arises from allowing the LeakFile query is that $\mathcal{A}$ can do offline password guessing using $\mathsf{k}_{\mathsf{OPRF}}$ obtained from leaked files. If $\mathcal{A}$ guesses the correct password pw of some cid, it can decrypt the ciphertext c that was encrypted under the output ρ of the OPRF. Hence, c has to be simulated such that it decrypts to the key K chosen randomly by $\mathcal{F}_{\mathsf{PPKR}}$ for cid, as otherwise, the simulation would be distinguishable from the real protocol. However, when we simulate c in the initialization phase, K is unknown, as $\mathcal{F}_{\mathsf{PPKR}}$ has not even given it to cid, yet. To solve this, we require AE to be equivocable, which allows us to produce a simulated c and only later decide to which values c decrypts. More precisely, whenever $\mathcal{A}$ queries H_2 on an input of the form $(\mathsf{pw}' \parallel \mathsf{cid}, H_1(\mathsf{pw}' \parallel \mathsf{cid})^{\mathsf{k}_{\mathsf{OPRF}}})$, where $\mathsf{k}_{\mathsf{OPRF}}$ is some OPRF key chosen by $\textsc{Sim}_{\mathsf{2HDH}}$, we submit pw' to the OfflineAttack interface of $\mathcal{F}_{\mathsf{PPKR}}$. If $\mathsf{pw} = \mathsf{pw}'$, then we obtains the key K and can equivocate c to obtain a ρ such that c decrypts to K under ρ and program the output of H_2 to ρ.

A similar challenge arises when S is fully corrupt. Then, S can make the critical query to H_2 even before we simulate c as it chooses the OPRF key $\mathsf{k}_{\mathsf{OPRF}}$ used in the initialization. Thus, we have to check for all previous H_2 queries whether it was this critical query via the OfflineAttack interface and if so, encrypt K under the corresponding output of H_2 instead of outputting an equivocable c.

The last major challenge is that a fully corrupt S can do key-planting attacks and can mix-and-match different files during a retrieval, e.g., when cid retrieves, S could use $\mathsf{k}_{\mathsf{OPRF}}$ of some $\mathsf{cid}' \neq \mathsf{cid}$ and an adversarial c. To deal with this, we check whether the password $\mathsf{pw}'' \parallel \mathsf{cid}''$ and key k'_{oprf} used to derive the ρ, under which the adversarial c was encrypted, are the same as the password $\mathsf{pw} \parallel \mathsf{cid}$ and key $\mathsf{k}_{\mathsf{OPRF}}$ used in the retrieval by cid. In the real world, cid would then output the key K decrypted from c. To simulate this, we use the key-planting capabilities in the RetS interface. We can again find the password $\mathsf{pw}'' \parallel \mathsf{cid}''$ used to derive ρ via the H_2 queries by $\mathcal{A}$ and check whether the same OPRF key was used with the help of $\textsc{Sim}_{\mathsf{2HDH}}$. We can then decrypt c to get K'' and submit pw'' and K'' to the RetS interface. If $\mathsf{pw} = \mathsf{pw}''$, cid then outputs K''. We can proceed similarly if c instead is equivocable.

□

We depict our simulator in Figures 5.11 to 5.14. In the below, when referring to blue-colored boxes such as I.1, we always mean the ones from these figures. The gray boxes CIS.4 refer to $\mathcal{F}_{\mathsf{PPKR}}$ instructions from Figures 4.7 and 4.8. The simulator works with session state records

$$\langle \textsc{Init}, \mathsf{cid}, \mathsf{kid}, \mathsf{ssid}, a, \mathsf{cid}^*, \mathsf{kid}, a^*, \mathsf{sk}_{\mathsf{enc}}, \mathsf{sk}_C, b, C \rangle$$

for initialization. All values are initialized to $\perp$ and potentially updated throught an initialization. At the end of an initialization by cid, the record has the following semantics.

$\textsc{Init}_1 \in \{\mathsf{cid}, \perp\}$ is either the cid of an honest client running initialization or it is set to $\perp$ if the client is corrupt or if the corrupt S impersonates cid;

$\textsc{Init}_2 \in \{\mathsf{kid}, \perp\}$ is either the number of initializations started by cid, as seen by the HSM, or it is set to $\perp$ if the client is corrupt or if the corrupt S impersonates cid;

$\textsc{Init}_3 = \mathsf{ssid}$ indicates the sub-session id used by cid;

$\textsc{Init}_4 \in \{a, \perp\}$ is either the a value sent by a honest cid, or $\perp$ if cid is corrupt or if the corrupt S impersonates cid.

$\textsc{Init}_5 \in \{\mathsf{cid}^*, \perp\}$ is either the client name, delivered to the HSM, or set to $\perp$ if the server is fully corrupt;

$\textsc{Init}_6 \in \{\mathsf{kid}, \perp\}$ is the number of initializations started by cid^*, as seen by the HSM, or set to $\perp$ if the server is fully corrupt;

$\textsc{Init}_7 \in \{a^*, \perp\}$ indicates the a value, delivered to the HSM, or set to $\perp$ if the server is fully corrupt;

$\textsc{Init}_8 \in \{\mathsf{sk}_{\mathsf{enc}}, \perp\}$ is the encryption secret key, generated by the HSM for the current sub-session, or set to $\perp$ if the server is fully corrupt;

$\textsc{Init}_9 \in \{\mathsf{sk}_C, \perp\}$ is the client's signature secret key either decrypted from c or simulated for an honest client. It is set to $\perp$ if it cannot be extracted from a corrupted initialization;

$\textsc{Init}_{10} = b$ is either the b value computed by the honest HSM or the value received by cid if S is fully corrupt;

$\textsc{Init}_{11} \in \{C, \perp\}$ is the ciphertext C computed by cid or $\perp$ if cid is corrupt.

Similarly, the simulator works with the sollowing session state records

$$\langle \textsc{Ret}, \mathsf{cid}, \mathsf{kid}, \mathsf{ssid}, a, match, \mathsf{cid}^*, \mathsf{kid}^*, a^*, \mathsf{pk}_C, b, c \rangle$$

in the retrieval. Again, all values are initialized to $\perp$ and potentially updated throughout a retrieval. At the end of the retrieval by cid, the record has the following semantics.

$\textsc{Ret}_1 \in \{\mathsf{cid}, \perp\}$ is either the cid of an honest client running retrieval or it is set to $\perp$ if the client is corrupt or if the corrupt S impersonates cid;

$\textsc{Ret}_2 \in \{\mathsf{kid}, \perp\}$ is either the number of initializations started by cid when cid starts this retrieval, as seen by the HSM, or it is set to $\perp$ if the client is corrupt or if the corrupt S impersonates cid;

$\textsc{Ret}_3 = \mathsf{ssid}$ indicates the sub-session id used by cid;

$\textsc{Ret}_4 \in \{a, \perp\}$ is either the a' value sent by a honest cid, or $\perp$ if cid is corrupt or if the corrupt S impersonates cid.

$\text{RET}_5 \in \{0,1\}$ indicates whether the password in the retrieval is correct

$\text{RET}_6 \in \{\mathsf{cid}^*, \bot\}$ is either the client name, delivered to the HSM, or set to $\bot$ if the server is fully corrupt;

$\text{RET}_7 \in \{\mathsf{kid}^*, \bot\}$ is the number of initializations started by cid^* when S receives the first message in this retrieval, as seen by the HSM, or set to $\bot$ if the server is fully corrupt;

$\text{RET}_8 \in \{a^*, \bot\}$ indicates the a' value, delivered to the HSM, or set to $\bot$ if the server is fully corrupt;

$\text{RET}_9 \in \{\mathsf{pk}_C, \bot\}$ is the signature public key used by the HSM in this retrieval, or set to $\bot$ if the server is fully corrupt;

$\text{RET}_{10} = b'$ is either the b value computed by the honest HSM or the value received by cid if S is fully corrupt;

$\text{RET}_{11} \in \{c\}$ is the ciphertext c sent by the honest HSM or $\bot$ if S is fully corrupt.

We construct a sequence of hybrid games $\mathbf{G}_0$ to $\mathbf{G}_{14}$ where we gradually change the real-world execution of the protocol OPRF-PPKR (interacting with the hybrid functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$) to reach the ideal-world execution, where the environment interacts with the simulator from Figures 5.11 to 5.14 and the ideal functionality $\mathcal{F}_{\mathsf{PPKR}}$. We write $\Pr[\mathbf{G}_i]$ to denote the probability that the environment outputs 1 in the hybrid game $\mathbf{G}_i$.

Game $\mathbf{G}_0$: Real world. This is the real world.

Game $\mathbf{G}_1$: Create simulator. In this game we create two new entities called the ideal functionality $\mathcal{F}$ and the simulator Sim. Initially, $\mathcal{F}$ just forwards the input of the dummy parties to Sim and outputs what Sim instructs it to output. In particular, $\mathcal{F}$ has interfaces $(\text{InitC}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw})$, $(\text{InitS}, \mathsf{ssid}, \mathsf{cid})$,$(\text{RetC}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}')$, and $(\text{RetS}, \mathsf{ssid}, \mathsf{cid})$ that just forward the input to Sim. Additionally, $\mathcal{F}$ has the corruption and attack interfaces LeakFile, FullyCorrupt, MaliciousInit, MaliciousRet, and OfflineAttack with the exact same code as in $\mathcal{F}_{\mathsf{PPKR}}$. Note, however, that Sim does not use these interfaces yet.

The simulator executes the code of all honest parties of the protocol internally on the input that it is provided by $\mathcal{F}$ and it internally runs the code of the hybrid functionality $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$. Additionally, it creates records exactly as $\text{Sim}_{\mathsf{OPRF\text{-}PPKR}}$ does, but at this point never uses the values from any of these records. Note that these are just syntactical changes and the protocol is still executed as in the real world. We have

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Simulator $\text{SIM}_{\text{OPRF-PPKR}}$, part 1

$\text{SIM}_{\text{OPRF-PPKR}}$ stores a list $\mathsf{kid}[\cdot]$. Initially $\text{SIM}_{\text{OPRF-PPKR}}$ executes $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.KGen}(1^\lambda)$ and stores $\langle PK, \mathsf{pk}, \mathsf{sk}\rangle$. Whenever it outputs a message towards some cid or from $\mathcal{F}_{\mathsf{HSM}}^{\text{OPRF-PPKR}}$ to a corrupt S, it attestates the message with sk.
Wait for X means that $\text{SIM}_{\text{OPRF-PPKR}}$ does not proceed to the next instruction before receiving X and meanwhile gives back activation to $\mathcal{A}$. Once it receives X, it first proceeds with the instructions on input X and then continues at the instruction, where it waited for X. If a record cannot be retrieved, the query is ignored. For brevity we omit session identifier sid from all inputs, outputs, and records. $(\mathbf{G}_1)$

On $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest cid

I.1 If $\mathsf{kid}[\mathsf{cid}]$ is undefined, set $\mathsf{kid}[\mathsf{cid}] := 0$. Otherwise, set $\mathsf{kid}[\mathsf{cid}] := \mathsf{kid}[\mathsf{cid}] + 1$. $(\mathbf{G}_3)$ // new OPRF key for each init

I.2 Give input $(\text{EVAL}, \mathsf{sid}, \mathsf{cid} \parallel \mathsf{kid}[\mathsf{cid}], \mathsf{ssid}, \mathsf{S}, 0)$ to $\mathcal{F}_{\mathsf{OPRF}}$ from cid. This triggers Step 5 of SIM_{2HDH}, which outputs $(\mathsf{sid}, \mathsf{cid}, \mathsf{ssid}, a)$. $(\mathbf{G}_3)$
Record $\langle \text{INIT}, \mathsf{cid}, \mathsf{kid}[\mathsf{cid}], \mathsf{ssid}, a, \bot, \bot, \bot, \bot, \bot, \bot, \bot\rangle$ and send $(\mathsf{ssid}, a, \mathsf{cid})$ as message from cid to S. $(\mathbf{G}_1)$ // Output of $\mathcal{F}_{\mathsf{OPRF}}$ (eval. of 0) ignored

On $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest S

I.3 If S honest, wait for $(\mathsf{ssid}, a, \mathsf{cid})$ from cid to S. $(\mathbf{G}_1)$ // See $(\mathsf{ssid}, a, \mathsf{cid})$ interface

On $(\text{INIT}, \mathsf{ssid}, a, \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of cid: // any cid, honest S

Ia.1 If S is honest, wait for $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_1)$

Ia.2 If cid is corrupt:
- If $\mathsf{kid}[\mathsf{cid}]$ is undefined, set $\mathsf{kid}[\mathsf{cid}] := 0$, otherwise set $\mathsf{kid}[\mathsf{cid}] := \mathsf{kid}[\mathsf{cid}] + 1$. Set $\mathsf{kid} := \mathsf{kid}[\mathsf{cid}]$. $(\mathbf{G}_3)$ // Ensure new oprf key. If cid is honest this is ensured in INITC
- Record $\langle \text{INIT}, \bot, \bot, \mathsf{ssid}, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot\rangle$ $(\mathbf{G}_1)$

Ia.3 If cid is honest, retrieve $\langle \text{INIT}, \mathsf{cid}, [\mathsf{kid}], \mathsf{ssid}, *, \bot, \bot, \bot, \bot, \bot, \bot, \bot\rangle$. $(\mathbf{G}_3)$

Ia.4 Give input $(\text{INIT}, \mathsf{sid}, \mathsf{cid} \parallel \mathsf{kid})$ to the simulated $\mathcal{F}_{\mathsf{OPRF}}$ from the simulated S. This triggers Step 2 of SIM_{2HDH}. $(\mathbf{G}_3)$ // Create new OPRF key

Ia.5 Give input $(\text{SNDRCOMPLETE}, \mathsf{sid}, \mathsf{cid} \parallel \mathsf{kid}, \mathsf{ssid})$ to the simulated $\mathcal{F}_{\mathsf{OPRF}}$ from the simulated S. This triggers Step 6 of SIM_{2HDH}, which outputs $(\mathsf{sid}, \mathsf{ssid}, b)$. $(\mathbf{G}_3)$

Ia.6 Compute $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\lambda)$ and overwrite the entries 6-11 in $\langle \text{INIT}, *, *, \mathsf{ssid}, *, *, *, *, *, *, *, *\rangle$ with $\mathsf{cid}, \mathsf{kid}, a, \mathsf{sk}_{\mathsf{enc}}, \bot, b$. Send $(\mathsf{ssid}, b, \mathsf{cid}, \mathsf{pk}_{\mathsf{enc}})$ as HSM-signed message from S to cid. $(\mathbf{G}_1)$ // Can later access OPRF keys via F records in SIM_{2HDH}

On $(\text{INIT}, \mathsf{ssid}, a, \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\text{OPRF-PPKR}}$ on behalf of corrupt S: // any cid

Ia.7 If no record $\langle \text{INIT}, *, [\mathsf{kid}], \mathsf{ssid}, *, *, *, *, *, *, *, *\rangle$ exists, create $\langle \text{INIT}, \bot, \bot, \mathsf{ssid}, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot\rangle$. $(\mathbf{G}_1)$
Additionally, if $\mathsf{kid}[\mathsf{cid}]$ is undefined set $\mathsf{kid}[\mathsf{cid}] := 0$ and otherwise $\mathsf{kid}[\mathsf{cid}] := \mathsf{kid}[\mathsf{cid}] + 1$ and set $\mathsf{kid} := \mathsf{kid}[\mathsf{cid}]$. $(\mathbf{G}_3)$ // cid corrupt and S just forwards or S impersonates cid

Ia.8 Execute Steps Ia.4 - Ia.6, except that the output is returned to S instead of sending it from S to cid. $(\mathbf{G}_3)$

Figure 5.11: Simulator $\text{SIM}_{\text{OPRF-PPKR}}$ for OPRF-PPKR, part 1.

Game $\mathbf{G}_2$: Output FAIL on tampered message from HSM. In this game, we change how the simulator reacts when an honest client receives a message from the HSM that was tampered with while the server is not fully corrupt. Whenever an honest client receives a message, on behalf of the HSM, where SIM never produced this message on behalf of the HSM, and if the sever is not fully corrupt, then SIM makes the client output FAIL .

The distribution of $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical, unless some honest client receives a message that was never simulated by SIM on behalf of the HSM, but where the attestation signature is still valid, which we denote as the event $\mathrm{E}_{\mathsf{Sig}}$. Assume there is some environment $\mathcal{Z}^*$ that is able to distinguish between $\mathbf{G}_2$ and $\mathbf{G}_1$. Then we construct an adversary $\mathcal{B}_1$ against the sEUF-CMA-security of Sig. $\mathcal{B}_1$ internally runs the whole experiment including $\mathcal{F}$, SIM and $\mathcal{Z}^*$. Whenever it simulates a message on behalf of the HSM it uses the signature oracle SIGN to compute the attestation signature. This implicitly programs the attestation key of the HSM to the one sampled in the sEUF-CMA experiment. As the key is sampled uniformly at random both in sEUF-CMA experiment and $\mathbf{G}_2$, this does not change the distribution of the key. When the event $\mathrm{E}_{\mathsf{Sig}}$ occurs, $\mathcal{B}_1$ outputs the corresponding message and signature to its challenger. It is easy to see that $\mathcal{B}_1$ wins the experiment whenever $\mathrm{E}_{\mathsf{Sig}}$ occurs. Hence, we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_1}(\lambda).$$

Game $\mathbf{G}_3$: Simulate the OPRF execution. In this game, we replace the 2HashDH protocol with its ideal execution, namely functionality $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$ of Figure 4.10 and simulator $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, N)$ of Figure 4.12. We consequently let the simulated parties use the interfaces of $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$, e.g., instead of computing $a := H_1(\mathsf{pw} \,\|\, \mathsf{cid})^r$, a party sends $(\textsc{Eval}, \mathsf{sid}, \mathsf{cid} \,\|\, \mathsf{kid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw} \,\|\, \mathsf{cid})$ to $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$, where for the first initialization by cid we have $\mathsf{kid} = 0$ and with each subsequent initialization by cid it is incremented by 1. Whenever a party is supposed to obtain a PRF value, the simulator calls the RCVCOMPLETE interface of $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$. Since in this ideal OPRF execution, OPRF keys are chosen by $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, N)$ (see Step 2 of Figure 4.12), the simulator takes these keys whenever it has to store a file. Finally, the 2HashDH protocol messages a, b sent by honest parties are now simulated by $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, N)$. The formal changes can be read from Figures 5.11 to 5.14, marked with $(\mathbf{G}_3)$.

This and the previous game are indistinguishable up to the advantage of distinguishing the real 2HashDH execution from the ideal execution of $\mathcal{F}^{\ell}_{\mathsf{OPRF}}$ and $\textsc{Sim}_{\mathsf{2HDH}}(\mathsf{sid}, H_1, H_2, N)$, which means we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \mathbf{Adv}^{\mathcal{F}_{\mathsf{OPRF}}}_{\mathsf{2HDH},\textsc{Sim}_{\mathsf{2HDH}},\mathcal{Z}}(\lambda).$$

.

Game $\mathbf{G}_4$: Abort upon ambiguous ciphertexts. We let the simulator abort if it obtains an adversarially-generated AE ciphertext c such that for two values k, k' from records $\langle F, \mathsf{S}, [\mathsf{kid}], k, * \rangle$, $\langle F, \mathsf{S}, [\mathsf{kid}'], k', * \rangle$ stored by the OPRF simulator $\textsc{Sim}_{\mathsf{2HDH}}$, c successfully decrypts under k and k', i.e., the output of $\mathsf{AE.Dec}$ is not $\perp$ (cf. IC.2 (c), IC.7 (c)(ii), and Rb.3 (a)).

This and the previous game are indistinguishable by the random-key robustness of AE. The reduction is straightforward, running on two challenge keys k, k': it randomly chooses two occasions where $\mathcal{F}_{\mathsf{OPRF}}$ samples $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x) \xleftarrow{\$} \{0,1\}^\ell$ and instead sets $F_{\mathsf{sid},\mathsf{S},\mathsf{kid}}(x) := k$ for the first one and $:= k'$ for the second. Note that this does not change the distribution as k, k' are uniformly random. Ciphertexts c passing the winning condition of the random-key robustness game can be detected by trial-decrypting with k and k'. We hence have

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq q_E^2 \mathbf{Adv}^{\mathsf{RKR}}_{\mathcal{B}_2,\mathsf{AE}}(\lambda).$$

Game $\mathbf{G}_5$: Honest cid fails upon malicious AE ciphertext. In this game, we modify how $\textsc{Sim}$ proceeds when receiving an adversarial message or honestly delivered message $(\mathsf{ssid}, b', c, \mathsf{cid})$ from $\mathcal{A}$ to some honest cid. $\textsc{Sim}$ checks whether all of the following conditions hold:

- c was not produced by $\textsc{Sim}$,
- There is no value $F_{\mathsf{sid},\mathsf{S},*}(\mathsf{pw}' \parallel \mathsf{cid})$ defined in $\mathcal{F}_{\mathsf{OPRF}}$ that successfully decrypts c,
- There is a value $F_{\mathsf{sid},\mathsf{S},[\mathsf{kid}]}(\mathsf{pw}' \parallel \mathsf{cid})$ defined in $\mathcal{F}_{\mathsf{OPRF}}$ that successfully decrypts c, but in this retrieval a malicious S did not use the OPRF key with kid kid.

Formally, $\textsc{Sim}$ executes these checks via the code in Rb.3 (a) and Rb.3 (b). If all conditions hold, then $\textsc{Sim}$ lets cid immediately output $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ instead of following the protocol as in $\mathbf{G}_4$. If any condition does not hold, it proceeds as in $\mathbf{G}_4$. Note that $\mathbf{G}_5$ only differs if cid outputs $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Succ})$ in $\mathbf{G}_4$ and $(\textsc{RetRes}, \mathsf{ssid}, \textsc{Fail})$ in $\mathbf{G}_5$, which we denote by the event E_{AE}. We now construct an adversary $\mathcal{B}_3$ that breaks the $\mathsf{INT\text{-}CTXT}$-security of AE given any environment $\mathcal{Z}$ that causes E_{AE}.

$\mathcal{B}_3$ acts like $\textsc{Sim}$ and chooses $i^* \xleftarrow{\$} \{1, \ldots, q_{\textsc{Init}}\}$. Let cid^* denote the cid that executes the i^*-th initialization. If cid^* is corrupt, $\mathcal{B}_3$ aborts. Otherwise, it does not compute c by encrypting (K, sk_C) under ρ^* but instead submits (K, sk_C) to its encryption oracle. Note that this implicitly programs ρ^* to the key k^* chosen by the $\mathsf{INT\text{-}CTXT}$ experiment. Since k^* is chosen uniformly at random and $\mathcal{F}_{\mathsf{OPRF}}$ chooses its outputs uniformly at random, this means that there is no change in the distribution of c.

In any subsequent retrieval by cid^* before cid^* executes another initialization, $\mathcal{B}_3$ checks the conditions listed above, and if they hold, it outputs the c'

Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$, part 2

On $(\mathsf{ssid}, b, \mathsf{cid}^*, \mathsf{pk}_{\mathsf{enc}})$ from $\mathcal{A}$ to cid: // honest cid, any S

Ib.1 Retrieve $\langle \text{INIT}, \mathsf{cid}, [\mathsf{kid}], \mathsf{ssid}, *, *, *, *, *, *, *, * \rangle$. $(\mathbf{G}_1)$

Ib.2 If S is not honest, give input $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ or if $\mathcal{F}_{\mathsf{PPKR}}$ ignores the input, continue below. $(\mathbf{G}_{10})$ // Ensures that we can produce output for cid

Ib.3 If HSM signature does not verify, send $(\text{COMPLETEINITC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

Ib.4 If $\mathsf{cid} \neq \mathsf{cid}^*$, send $(\text{COMPLETEINITC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$ // cid's a rerouted

Ib.5 Compute $(\mathsf{sk}_C, \mathsf{pk}_C) \coloneqq \mathsf{Sig.KGen}(1^\lambda)$. $(\mathbf{G}_1)$

Ib.6 If S is fully corrupt:

a) Send $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

b) For each record $\langle H_2, [\mathsf{pw}] \parallel \mathsf{cid}, *, [y] \rangle$ marked CONSISTENT $(\mathbf{G}_{14})$ send $(\text{OFFLINEATTACK}, \mathsf{cid}, \mathsf{pw}, i+1)$ to $\mathcal{F}_{\mathsf{PPKR}}$, where $i \in \mathbb{N}$ is the largest number such that a record $\langle \text{LEAKED}, \mathsf{cid}, *, *, *, i \rangle$ exists. $(\mathbf{G}_{10})$

c) If for any query the output is $K \neq \text{FAIL}$ $(\mathbf{G}_{10})$, compute $c \xleftarrow{\$} \mathsf{AE.Enc}(y, (K, \mathsf{sk}_C))$. Else set $c \xleftarrow{\$} \text{SIM}_{\mathsf{EQV}}(\lambda + |\mathsf{sk}_C|)$. $(\mathbf{G}_6)$

d) Record $\langle \text{FILE}, \mathsf{cid}, \mathsf{kid}, \mathsf{pk}_C, \mathsf{sk}_C, c \rangle$ and $\langle \text{LEAKED}, \mathsf{cid}, \mathsf{kid}, \mathsf{sk}_C, c, i+1 \rangle$, where $i \in \mathbb{N}$ is the largest number such that a record $\langle \text{LEAKED}, \mathsf{cid}, *, *, *, i \rangle$ exists or $i = 0$ if no such record exists. $(\mathbf{G}_1)$

e) Compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, (\mathsf{ssid}, \mathsf{pk}_C, c))$. $(\mathbf{G}_1)$

f) Overwrite the three last entries of the INIT record retrieved above with sk_C, b, C. $(\mathbf{G}_1)$

Ib.7 If S is not fully corrupt:

a) Compute $c \xleftarrow{\$} \text{SIM}_{\mathsf{EQV}}(\lambda + |\mathsf{sk}_C|)$. $(\mathbf{G}_6)$

b) Compute $C \coloneqq \mathsf{PKE.Enc}(\mathsf{sk}_{\mathsf{enc}}, \bot)$, $(\mathbf{G}_{13})$

c) overwrite the tenth entry of the record retrieved above with sk_C and the last entry with C. $(\mathbf{G}_1)$

Ib.8 Send (ssid, C) to S $(\mathbf{G}_1)$ and $(\text{COMPLETEINITC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$. Store $\langle C, \mathsf{ssid}, \mathsf{pk}_C, c \rangle$ $(\mathbf{G}_{13})$.

On (ssid, C) from $\mathcal{A}$ to S on behalf of cid: // any cid, honest S

IC.1 Retrieve $\langle \text{INIT}, *, *, \mathsf{ssid}, *, *, [\mathsf{kid}], *, [\mathsf{sk}_{\mathsf{enc}}], [\mathsf{sk}_C], *, [C'] \rangle$. $(\mathbf{G}_1)$

IC.2 If cid is corrupt: $(\mathbf{G}_8)$

a) Execute HSM code, $(\mathbf{G}_1)$ where the entry $\mathsf{k}_{\mathsf{OPRF}}$ of the stored FILE record is set to $\bot$, $(\mathbf{G}_3)$ up to determining *out* and denote the decryption by $(\mathsf{ssid}', \mathsf{pk}_C, c)$. $(\mathbf{G}_1)$ If $out = \text{FAIL}$, send $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ and otherwise continue. $(\mathbf{G}_{11})$ // $\text{SIM}_{\mathsf{2HDH}}$ has the OPRF key

b) Find a record $\langle H_2, [\mathsf{pw}] \parallel *, *, [y] \rangle$ that is marked CONSISTENT such that $\bot \neq (K, \mathsf{sk}_C) \coloneqq \mathsf{AE.Dec}(y, c)$. Otherwise set $\mathsf{pw} \coloneqq \bot$. $(\mathbf{G}_8)$

c) If more than one consistent record is found, abort the simulation. $(\mathbf{G}_4)$

d) Send input $(\text{INITC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of cid. $(\mathbf{G}_8)$ On response $(\text{INITC}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ send message $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{11})$.

IC.3 If cid is honest, retrieve $\langle C, \mathsf{ssid}, [\mathsf{pk}_C], [c] \rangle$. $(\mathbf{G}_{13})$ Store $\langle \text{FILE}, \mathsf{cid}, \mathsf{pk}_C, c, \bot, 10 \rangle$ overwriting any $\langle \text{FILE}, \mathsf{cid}, *, *, *, * \rangle$ $(\mathbf{G}_1)$, and send message $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

Figure 5.12: Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ for OPRF-PPKR, part 2.

received in that retrieval to the INT-CTXT experiment. Further, if the file from the i^*-th initialization was leaked, in any subsequent retrieval by cid^*, $\mathcal{B}_3$ checks the same conditions and outputs the c' received in that retrieval to the INT-CTXT experiment if they hold.

If such a retrieval by cid^* is the first that causes the event E_{AE}, which happens with probability $1/q_{\text{INIT}}$, $\mathcal{B}_3$ wins the INT-CTXT experiment due to the following. Let c^* be the ciphertext that causes E_{AE}, which means that in $\mathbf{G}_4$, cid^* would successfully decrypt c^* using ρ^*. Since $\mathcal{B}_3$ implicitly set $\rho^* = k^*$, c^* successfully decrypts under k^* and constitutes a forgery in the INT-CTXT experiment. Therefore, we have

$$|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| \leq q_{\text{INIT}} \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{B}_3,\mathsf{AE}}(\lambda).$$

Game $\mathbf{G}_6$: Simulate ciphertexts of honest initializations without passwords. In this game, we change how SIM computes the AE ciphertext c in initializations of honest clients. Now, when SIM receives a message $(\mathsf{ssid}, b, \mathsf{cid}, \mathsf{pk}_{\mathsf{enc}})$ and $\mathcal{A}$ has not yet queried $(\mathsf{pw} \parallel \mathsf{cid}, b^{1/r})$ to the random oracle H_2, SIM does not compute $c \xleftarrow{\$} \mathsf{AE.Enc}(\rho, (K, \mathsf{sk}_C))$. Instead it uses the equivocability simulator of AE and runs $c \xleftarrow{\$} \text{SIM}_{\mathsf{EQV}}(\lambda + |\mathsf{sk}_C|)$. Note that this can only happen with non-negligible probability if S is fully corrupt as otherwise $\mathcal{A}$ would have to guess the OPRF key used by $\text{SIM}_{\mathsf{2HDH}}$. Then, SIM records $\langle \mathsf{AE}, c, \mathsf{pw} \parallel \mathsf{cid}, b^{1/r}, K, \mathsf{sk}_C \rangle$. Recall that SIM still receives pw as input and can create these records. We emphasize that the creation of these records is only a temporary change that will be removed later in $\mathbf{G}_{14}$ and any step introduced in this game that relies on these records will be changed later to work without them (cf. $\mathbf{G}_{10}$ and $\mathbf{G}_{14}$).

We now use the AE records to ensure that retrieval still works correctly. To that end, we change SIM such that when it receives a message $(\mathsf{ssid}, b', c, \mathsf{cid})$ in a retrieval of an honest cid, SIM retrieves the record $\langle \mathsf{AE}, c, \mathsf{pw}' \| \mathsf{cid}, b'^{1/r'}, [K], [\mathsf{sk}_C] \rangle$. If such a record exists, c was produced $\text{SIM}_{\mathsf{EQV}}$, and SIM proceeds using K and sk_C from that record. If no such record exists, then c was computed by $\mathcal{Z}$, and SIM checks whether cid can decrypt it. For this, it again checks the conditions from $\mathbf{G}_5$. If they hold, SIM let's cid output FAIL and otherwise decrypts c using $F_{\mathsf{sid},\mathsf{S},\mathsf{cid}\|*}(\mathsf{pw}' \parallel \mathsf{cid})$ to obtain K and sk_C. Note that the output behavior of cid here is the same as in $\mathbf{G}_5$. Formally, all these changes that do not rely on the AE records are done in Ib.6 (c) and Ib.7 (a).

To ensure that $\mathcal{Z}$ cannot distinguish $\mathbf{G}_6$ from $\mathbf{G}_5$, SIM additionally has to properly program the random oracle H_2 to account for password guessing. For each query $(\mathsf{pw} \parallel \mathsf{cid}, y)$ to H_2, it retrieves the record $\langle \mathsf{AE}, [c], \mathsf{pw} \parallel \mathsf{cid}, y, [K], [\mathsf{sk}_C] \rangle$, then runs $\rho \coloneqq \text{SIM}_{\mathsf{EQV}}(c, (K, \mathsf{sk}_C))$, and sets $H_2(\mathsf{pw}, y) \coloneqq \rho$ (H2.2 (b)). That means, whenever $\mathcal{A}$ queries H_2 on the input that would yield the key for c in $\mathbf{G}_5$, SIM now equivocates c to the matching message

and programs H_2 accordingly. If no such record exists, SIM forwards the query to the random oracle H_2 of $\text{SIM}_{\mathsf{2HDH}}$ as in $\mathbf{G}_5$.

We claim that for any $\mathcal{Z}$ that is able to distinguish $\mathbf{G}_6$ and $\mathbf{G}_5$, we can construct an adversary $\mathcal{B}_4$ that wins the EQV experiment for AE. Since in $\mathbf{G}_5$, ρ is chosen by $\mathcal{F}_{\mathsf{OPRF}}$ uniformly at random and c is encrypted under ρ, the distribution of ρ and c in $\mathbf{G}_5$ is exactly the same as in the real game of the EQV experiment. In $\mathbf{G}_6$, c is output by $\text{SIM}_{\mathsf{EQV}}$ and thus its distribution is exactly the same as in the ideal game of the EQV experiment. The distribution of ρ in $\mathbf{G}_6$ remains unchanged unless there is a query to H_2 such that SIM equivocates c. In that case ρ is distributed exactly the same as in the ideal world in the EQV experiment as it is computed by $\text{SIM}_{\mathsf{EQV}}$. Thus, we have

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \mathbf{Adv}^{\text{EQV}}_{\mathcal{B}_4,\mathsf{AE}}(\lambda).$$

Game $\mathbf{G}_7$: Abort upon signature forgery. In this game, we let SIM abort upon observing a forged signature. Concretely, if SIM receives a message $(\mathsf{ssid}, \sigma^*)$ from $\mathcal{A}$ to S on behalf of cid then SIM aborts if all of the following conditions hold (cf. $\sigma.1$):

- There is a record $\langle \text{FILE}, \mathsf{cid}, [\mathsf{pk}_C], *, *, * \rangle$ such that $\mathsf{Sig.Vrfy}(\mathsf{pk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c), \sigma^*) = 1$,
- The signed c was not equivocated using $\text{SIM}_{\mathsf{EQV}}$, i.e., SIM never computed $\rho := \text{SIM}_{\mathsf{EQV}}(c, (K, \mathsf{sk}_C))$,
- σ^* was not computed by SIM on behalf of an honest cid.

Similarly, if SIM receives a message $(\mathsf{ssid}, \sigma^*)$ from $\mathcal{A}$ to $\mathcal{F}^{\mathsf{OPRF\text{-}PPKR}}_{\mathsf{HSM}}$ on behalf of S then SIM aborts if all of the above conditions hold (cf. $\sigma.2$). We call this event E_{Sig}. If $\mathcal{A}$ can provoke E_{Sig} to happen, then we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}_5$ against the $\mathsf{sEUF\text{-}CMA}$ security of Sig. $\mathcal{B}_5$ internally runs $\mathcal{Z}$ and plays the role of SIM and $\mathcal{F}$ in the execution of the protocol. $\mathcal{B}_5$ starts by randomly choosing $i^* \in \{1, \ldots, q_{\text{INIT}}\}$, where q_{INIT} is the number of initializations. Let cid^* denote the cid that exeutes the i^*-th initialization. If cid^* is honest, then $\mathcal{B}_5$ does not generate a signing key pair $(\mathsf{sk}_C, \mathsf{pk}_C)$ but instead uses the public key pk^* that it gets from its challenger. Since it does not know the key sk^*, it stores $\perp$ instead of sk_C in the AE record in the i^*-th initialization.

If cid^* is corrupt, then $\mathcal{B}_5$ aborts. Further, if $\mathcal{A}$ makes an H_2 query such that SIM would equivocate c^* by computing $\rho := \text{SIM}_{\mathsf{EQV}}(c^*, (K, \mathsf{sk}_C))$, where c^* is the AE ciphertext produced in the i^*-th initialization, then $\mathcal{B}_5$ also aborts.

Now, whenever cid^* executes a retrieval, $\mathcal{B}_5$ checks two conditions:

- cid^* receives a message $(\mathsf{ssid}, b', c^*, \mathsf{cid}^*)$, where c^* is the ciphertext it produced in the i^*-th initialization.
- The retrieval is successful (in this game, $\mathcal{B}_5$ still gets the client input pw' and can check $\mathsf{pw} = \mathsf{pw}'$).

Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$, part 3.

On (ssid, C) from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ on behalf of corrupt S: // any cid, corrupt S

IC.4 Retrieve $\langle \text{INIT}, [\mathsf{cid}], *, \mathsf{ssid}, [a], [\mathsf{cid}^*], [\mathsf{kid}], [a^*], [\mathsf{sk}_{\mathsf{enc}}], [\mathsf{sk}_C], *, [C'] \rangle$. $(\mathbf{G}_1)$

IC.5 If $C' = \perp$, give input $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{INITS}, \mathsf{ssid}, \mathsf{cid})$ from $\mathcal{F}_{\mathsf{PPKR}}$ continue below. $(\mathbf{G}_{10})$ // $C' \neq \perp$ implies that b was deliverd to cid, where the simulator gave input INITS to $\mathcal{F}_{\mathsf{PPKR}}$

IC.6 If $C = C'$, retrieve record $\langle C, [\mathsf{ssid}], [\mathsf{pk}_C], [c] \rangle$ $(\mathbf{G}_{13})$. Give input $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{10})$,
record $\langle \text{FILE}, \mathsf{cid}^*, \mathsf{kid}, \mathsf{pk}_C, \mathsf{sk}_C, c \rangle$ overwriting any $\langle \text{FILE}, \mathsf{cid}^*, *, *, *, * \rangle$, and send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to S as output of the HSM. $(\mathbf{G}_1)$ // honest delivery of C, which also implies honest delivery of a and $\mathsf{cid} = \mathsf{cid}^*$ as otherwise $C' = \perp$

IC.7 If $C \neq C'$:

a) Search for a record $\langle C, [\mathsf{ssid}'], [\mathsf{pk}_C], [c] \rangle$. $(\mathbf{G}_{13})$
If $\mathsf{ssid}' \neq \mathsf{ssid}$, send $(\text{INITRES}, \mathsf{ssid}, \text{FAIL})$ to S. Otherwise, record $\langle \text{FILE}, \mathsf{cid}^*, \mathsf{kid}, \mathsf{pk}_C, \mathsf{sk}_C, c \rangle$ overwriting any $\langle \text{FILE}, \mathsf{cid}^*, *, *, *, * \rangle$. $(\mathbf{G}_1)$

b) If no such record $\langle C, *, *, * \rangle$ exists $(\mathbf{G}_{13})$, execute the HSM code on input (ssid, C), $(\mathbf{G}_1)$
where the entry $\mathsf{k}_{\mathsf{OPRF}}$ of the stored FILE record is set to $\perp$, $(\mathbf{G}_3)$
up to determining *out* and denote the result of the decryption by $(\mathsf{ssid}', \mathsf{pk}_C, c)$. If *out* = FAIL, send $(\text{INITRES}, \mathsf{ssid}, \text{FAIL})$ to S. If *out* $\neq$ FAIL, record $\langle \text{FILE}, \mathsf{cid}^*, \mathsf{kid}, \mathsf{pk}_C, \mathsf{sk}_C, c \rangle$ overwriting any $\langle \text{FILE}, \mathsf{cid}^*, *, *, *, * \rangle$. $(\mathbf{G}_1)$

c) Search a record $\langle H_2, [\mathsf{pw}] \parallel *, *, [y] \rangle$ that is marked CONSISTENT such that $\perp \neq (K, \mathsf{sk}_C) := \mathsf{AE.Dec}(y, c)$. Overwrite the third to last entry of the RET record retrieved above with sk_C. $(\mathbf{G}_8)$

i. If no such record exists, send $(\text{MALICIOUSINIT}, \mathsf{cid}^*, \perp, \perp)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_8)$ // $\mathcal{Z}$ has not yet computed the PRF value for a

ii. If more than one consistent record is found, abort the simulation. $(\mathbf{G}_4)$

iii. In all other cases, send $(\text{MALICIOUSINIT}, \mathsf{ssid}, \mathsf{cid}^*, \mathsf{pw}, K)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_8)$ // $\mathcal{Z}$ can decrypt c

d) Send $(\text{INITRES}, \mathsf{ssid}, \text{SUCC})$ to S as output of the HSM. $(\mathbf{G}_1)$

On random oracle query $H_1(\mathsf{pw} \parallel \mathsf{cid})$ from $\mathcal{A}$:

H1.1 Query $H_1(\mathsf{pw} \parallel \mathsf{cid})$ to $\text{SIM}_{\mathsf{2HDH}}$ ($\rightarrow$ Step 4 of $\text{SIM}_{\mathsf{2HDH}}$). Return output. $(\mathbf{G}_3)$

On random oracle query $H_2(\mathsf{pw} \parallel \mathsf{cid}, x)$ from $\mathcal{A}$:

H2.1 If a record $\langle H_2, \mathsf{pw} \parallel \mathsf{cid}, x, y \rangle$ exists, output y. $(\mathbf{G}_1)$

H2.2 If there exists a record $\langle F, \mathsf{S}, \mathsf{cid} \| [\mathsf{kid}], [\mathsf{k}_{\mathsf{OPRF}}], * \rangle$ in $\text{SIM}_{\mathsf{2HDH}}$ such that $x = H_1(\mathsf{pw} \| \mathsf{cid})^{\mathsf{k}_{\mathsf{OPRF}}}$: $(\mathbf{G}_{14})$

a) Retrieve $\langle \text{LEAKED}, \mathsf{cid}, \mathsf{kid}, [\mathsf{sk}_C], [c], [i] \rangle$. If multiple exist, choose smallest i. $(\mathbf{G}_{10})$

b) Send $(\text{OFFLINEATTACK}, \mathsf{cid}, \mathsf{pw}, i)$ to $\mathcal{F}_{\mathsf{PPKR}}$. If it outputs $K \neq \text{FAIL}$, $(\mathbf{G}_{10})$
run $y \xleftarrow{\$} \text{SIM}_{\mathsf{EQV}}(c, (K, \mathsf{sk}_C))$, record $\langle H_2, \mathsf{pw} \parallel \mathsf{cid}, x, y \rangle$ and output y. $(\mathbf{G}_6)$
Otherwise continue.

H2.3 Query $H_2(\mathsf{pw} \| \mathsf{cid}, x)$ to $\text{SIM}_{\mathsf{2HDH}}$, which triggers Step 8 of $\text{SIM}_{\mathsf{2HDH}}$. Let y denote its output. Record $\langle H_2, \mathsf{pw} \parallel \mathsf{cid}, x, y \rangle$. If y was produced by $\mathcal{F}_{\mathsf{OPRF}}$ mark the record as CONSISTENT. Output y. $(\mathbf{G}_3)$

Figure 5.13: Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ for OPRF-PPKR, part 3.

If both conditions are satisfied, then $\mathcal{B}_5$ does not compute $\mathsf{Sig.Sign}(\mathsf{sk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c^*))$ but instead uses its $\mathsf{Sign}(\cdot)$ oracle provided by its challenger to get a signature σ on the message $m = (a', \mathsf{cid}, \mathsf{ssid}, b', c^*)$. If any condition does not hold, $\mathcal{B}_5$ proceeds just like Sim in $\mathbf{G}_6$. Now, $\mathcal{B}_5$ continues the simulation and observes all (ssid, σ) messages that it receives from $\mathcal{A}$. If there is a message such that $\mathrm{E}_{\mathsf{Sig}}$ happens, then $\mathcal{B}_5$ outputs σ.

Note that the view of $\mathcal{Z}$ in the reduction did not change with respect to $\mathbf{G}_7$. The public key pk_C and the signatures σ are distributed exactly as in $\mathbf{G}_6$. In the i^*-th initialization, $\mathcal{B}_5$ did not use sk_C, as it simulated $c^* \xleftarrow{\$} \textsc{Sim}_{\mathsf{EQV}}(\lambda + |\mathsf{sk}_C|)$ without sk_C as in $\mathbf{G}_6$. $\mathcal{B}_5$ also did *not* have to equivocate c^* with $\rho := \textsc{Sim}_{\mathsf{EQV}}(c^*, (K, \mathsf{sk}_C))$ (which it would not be able to do). If the first signature σ^* that caused the event $\mathrm{E}_{\mathsf{Sig}}$ is valid under pk^*, which happens with probability at least $1/q_{\textsc{Init}}$, $\mathcal{B}_5$ wins. Overall, we get

$$|\Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6]| \leq q_{\textsc{Init}} \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{B}_5, \mathsf{Sig}}(\lambda).$$

Game $\mathbf{G}_8$: Extract from malicious initialization. In this game, we change the behaviour of Sim whenever it receives a message (ssid, C) from a corrupt cid or the corrupt S queries (ssid, C) to $\mathcal{F}^{\mathsf{OPRF\text{-}PPKR}}_{\mathsf{HSM}}$. For this, we first modify $\mathcal{F}$ in the following way. We change the interface $(\textsc{InitC}, \mathsf{ssid}, \mathsf{pw})$ such that it acts exactly as $\mathcal{F}_{\mathsf{PPKR}}$ if and only if the cid that makes the query is corrupt. Further, we add an interface $(\textsc{CompleteInitS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K)$ that executes the steps CIS.3 and CIS.4 of $\mathcal{F}_{\mathsf{PPKR}}$. Again, the addition of this interface is only a temporary change and it is removed again in $\mathbf{G}_{11}$.

When receiving a message (ssid, C) from a corrupt party, Sim executes the protocol as in $\mathbf{G}_7$ and then searches for a record $\langle H_2, \mathsf{pw} \,\|\, *, *, y\rangle$ such that c decrypts successfully to some (K, sk_C) under y. Note that as of game $\mathbf{G}_4$, there is at most one such y. If no such record exists, it sets $K := \bot, \mathsf{pw} := \bot$.

If the message (ssid, C) came from a corrupt cid, Sim gives the input $(\textsc{InitC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of cid and sends $(\textsc{CompleteInitS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On the other hand, if (ssid, C) came from the corrupt S, Sim queries $(\textsc{MaliciousInit}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K)$ to $\mathcal{F}_{\mathsf{PPKR}}$, where cid is the value the corrupt S sent in the first message of the subsession ssid. Formally, the changes can be read from IC.2 (Step IC.2 (b) and the first part of Step IC.2 (d)) and IC.7 (c) (Steps IC.7 (c)(i) and IC.7 (c)(iii)).

The changes introduced in this game are only syntactical and do not affect any output of Sim. Essentially we only let $\mathcal{F}$ store some records for initializations executed by a corrupt party, but $\mathcal{F}$ never uses these records yet to produce an output for any party. Hence, we have

$$\Pr[\mathbf{G}_8] = \Pr[\mathbf{G}_7].$$

Game $\mathbf{G}_9$: Extract from malicious retrieval. We change the simulator whenever it receives message (ssid, σ) from a corrupt cid to S, or from a corrupt S to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$. As of game $\mathbf{G}_7$, we know that if σ verifies under pk_C stored in the file of cid, either the adversary previously initialized (on behalf of corrupt cid or a corrupt S) that file record or it guessed the password of cid. As in the previous game, we let the simulator extract pw from that malicious initialization or password guess and submit $(\textsc{RetC}, \mathsf{ssid}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of corrupt cid (see $\sigma.2$ (a) and $\sigma.2$ (b)), resp. $(\textsc{MaliciousRet}, \mathsf{cid}^*, \mathsf{pw}')$ for a corrupt S (see $\sigma.16$).

Because the changes in the simulation again only affect the state of $\mathcal{F}_{\mathsf{PPKR}}$ which is not yet influencing protocol outputs, the change is only syntactical, and we have

$$Pr[\mathbf{G}_9] = \Pr[\mathbf{G}_8].$$

Game $\mathbf{G}_{10}$: Let $\mathcal{F}$ produce the output of the client. In this game, we change the simulator such that the output for honest clients is generated by $\mathcal{F}$. To this end we have to introduce several changes to $\mathcal{F}$. We modify the interfaces $(\textsc{InitC}, \mathsf{ssid}, \mathsf{pw})$ and $(\textsc{RetC}, \mathsf{ssid}, \mathsf{pw})$ to act exactly like $\mathcal{F}_{\mathsf{PPKR}}$, except that they still forward all inputs to Sim. Next, we add the interfaces $(\textsc{InitS}, \mathsf{ssid}, \mathsf{cid})$, $(\textsc{RetS}, \mathsf{ssid}, \mathsf{cid})$, $(\textsc{CompleteInitC}, \mathsf{ssid}, b_C)$, and $(\textsc{CompleteRetC}, \mathsf{ssid}, b_C)$ and let them act exactly like in $\mathcal{F}_{\mathsf{PPKR}}$. Finally, we introduce the interfaces $(\textsc{CompleteInitS}, \mathsf{ssid}, b_S)$ and $(\textsc{CompleteRetS}, \mathsf{ssid}, b_S)$, which both act as in $\mathcal{F}_{\mathsf{PPKR}}$, except that they never give any output to S.

Furthermore, we change Sim to always use the interfaces CompleteInitC and CompleteRetC of $\mathcal{F}$ whenever it wants to produce output of some honest cid with b_C set appropriately (Ib.3, Ib.4, Ib.8, Del.3, Rb.5, Rb.6, Rb.8, last step, and Rb.9, last step). Note that these interfaces only produce output if both InitC and InitS, resp. RetC and RetS, were queried previously and thus require Sim to give the inputs to $\mathcal{F}$ on behalf of S if S is not honest (Ib.2, Del.2, Rb.4). Additionally, Sim has to ensure that $\mathcal{F}$ internally creates File records that can be accessed during retrievals. For this reason, whenever $\mathcal{A}$ sends some (ssid, C) from an honest cid, Sim sends $(\textsc{CompleteInitS}, \mathsf{ssid}, 1)$ to $\mathcal{F}$ (IC.3, IC.6), once again first giving the InitS input to $\mathcal{F}$ if S is not honest to ensure that the CompleteInitS query proceeds (IC.5). Lastly, Sim acts exactly like $\text{Sim}_{\mathsf{OPRF\text{-}PPKR}}$ on the queries LeakFile and $(\textsc{FullyCorrupt}, \mathsf{S})$ by $\mathcal{A}$ (LF.1 - LF.3, FC.1 - FC.3).

A difficulty for Sim resulting from these changes is that for honest clients, K is now chosen by $\mathcal{F}$ instead of Sim and is unknown to Sim. In particular, this affects the changes introduced in $\mathbf{G}_6$ as Sim cannot use the records $\langle \mathsf{AE}, \ldots \rangle$ anymore in $\mathbf{G}_{10}$ to store and obtain K. Instead, whenever Sim equivocates some ciphertext c during a query to H_2, it has to extract K from $\mathcal{F}$. Note

Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$, part 4.

On $(\text{RETC}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$ // Any cid

R.1 If no record $\langle \text{FILE}, \mathsf{cid}, [\mathsf{kid}], *, *, * \rangle$ exists, set $\mathsf{kid} := \bot$. $(\mathbf{G}_3)$

R.2 Give input $(\text{EVAL}, \mathsf{sid}, \mathsf{cid} \parallel \mathsf{kid}, \mathsf{ssid}, \mathsf{S}, 0)$ to the simulated $\mathcal{F}_{\mathsf{OPRF}}$ from the simulated cid. This triggers Step 5 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{cid}, \mathsf{ssid}, a')$. $(\mathbf{G}_3)$

R.3 If a record $\langle \text{RET}, \bot, \bot, \mathsf{ssid}, \bot, \bot, *, *, *, *, *, * \rangle$ exists, overwrite the second, third, fifth, and sixth entry with $(\mathsf{cid}, \mathsf{kid}, a', \mathit{match})$. $(\mathbf{G}_1)$ // This record exists if RETS was executed previously

R.4 Otherwise, record $\langle \text{RET}, \mathsf{cid}, \mathsf{kid}, \mathsf{ssid}, a', \mathit{match}, \bot, \bot, \bot, \bot, \bot, \bot \rangle$. $(\mathbf{G}_1)$

R.5 Send $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ as message from cid to S. $(\mathbf{G}_1)$

On $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$: // honest, corrupt, or fully corrupt S

R.6 If a record $\langle \text{RET}, *, *, \mathsf{ssid}, *, \bot, *, *, *, *, *, * \rangle$ exists, overwrite $\bot$ with match. $(\mathbf{G}_1)$ // This record exists if RETC was executed previously

R.7 Otherwise, record $\langle \text{RET}, \bot, \bot, \mathsf{ssid}, \bot, \mathit{match}, \bot, \bot, \bot, \bot, \bot, \bot \rangle$. $(\mathbf{G}_1)$

R.8 Wait for $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$. $(\mathbf{G}_1)$ // See the $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ interface below.

On $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ from $\mathcal{A}$ to S on behalf of cid: // any cid, any (if jumping here from the $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ interface) S

Ra.1 If S is honest, wait for $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ message from $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_1)$ // See $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ interface.

Ra.2 Retrieve $\langle \text{FILE}, \mathsf{cid}, [\mathsf{kid}], [\mathsf{pk}_C], *, [c'] \rangle$. If no such record exists and S is honest, send $(\text{COMPLETERETS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{12})$

Ra.3 Execute HSM code on input $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ up to determining the output $(\mathbf{G}_1)$ and with the computation of b' substituted as follows. Give input $(\text{SNDRCOMPLETE}, \mathsf{sid}, \mathsf{cid} \parallel \mathsf{kid}, \mathsf{ssid})$ to the simulated $\mathcal{F}_{\mathsf{OPRF}}$ from the simulated S. This triggers Step 6 of $\text{SIM}_{\mathsf{2HDH}}$, which outputs $(\mathsf{sid}, \mathsf{ssid}, b')$. $(\mathbf{G}_3)$

Ra.4 If the HSM output is $\mathsf{ssid}, \text{DELREC}$, record $\langle \text{DELREC}, \mathsf{ssid} \rangle$ and send $(\mathsf{ssid}, \text{DELREC})$ as HSM-signed message from S to cid. If the output is $\mathsf{ssid}, b', c, \mathsf{cid}$, overwrite the six last entries of $\langle \text{RET}, *, *, \mathsf{ssid}, *, *, *, *, *, *, *, * \rangle$ with $\mathsf{cid}, \mathsf{kid}, a', \mathsf{pk}_C, b', c'$ and send $(\mathsf{ssid}, b', c', \mathsf{cid})$ as HSM-signed message to cid. $(\mathbf{G}_1)$

On $(\text{RET}, \mathsf{ssid}, a', \mathsf{cid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ on behalf of corrupt S:

Ra.5 If no record $\langle \text{RET}, *, *, \mathsf{ssid}, *, *, *, *, *, *, *, * \rangle$ exists, create $\langle \text{RET}, \bot, \bot, \mathsf{ssid}, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot \rangle$. $(\mathbf{G}_1)$

Ra.6 Execute Steps Ra.2 - Ra.4 except that the output is returned to S instead of sending it from S to cid $(\mathbf{G}_3)$

On $(\mathsf{ssid}, \text{DELREC})$ from $\mathcal{A}$ to cid on behalf of S: // honest cid, any S

Del.1 If no record $\langle \text{RET}, \mathsf{cid}, *, \mathsf{ssid}, *, *, *, *, *, *, *, * \rangle$ exists or if there already was a message $(\mathsf{ssid}, b', c, \mathsf{cid}^*)$ to cid, ignore. $(\mathbf{G}_1)$

Del.2 If S is not honest, send $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, 0, \text{DELREC}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}, \mathit{match})$ from $\mathcal{F}_{\mathsf{PPKR}}$ continue below. $(\mathbf{G}_{10})$

Del.3 If HSM signature verifies, send $(\text{COMPLETERETC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ and otherwise $(\text{COMPLETERETC}, \mathsf{ssid}, \mathsf{cid}, 0)$. $(\mathbf{G}_{10})$

Figure 5.14: Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ for OPRF-PPKR, part 4.

Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$, part 5.

On $(\mathsf{ssid}, b', c, \mathsf{cid}^*)$ from $\mathcal{A}$ to cid on behalf of S: // honest cid, any S

Rb.1 Retrieve $\langle \text{RET}, \mathsf{cid}, *, \mathsf{ssid}, [a'], [match], *, [\mathsf{kid}], *, *, *, * \rangle$. $(\mathbf{G}_1)$
If no such record exists or if there already was a message $(\mathsf{ssid}, \text{DELREC})$ to cid, ignore. $(\mathbf{G}_1)$ // cid did not start retrieval or record deleted.

Rb.2 If S is corrupt, set $\mathsf{pw} \coloneqq \bot, K \coloneqq \bot, i \coloneqq \bot$. $(\mathbf{G}_{12})$

Rb.3 If S is fully corrupt:

a) If record $\langle H_2, [\mathsf{pw}] \parallel \mathsf{cid}, [u], [y] \rangle$ marked CONSISTENT exists $(\mathbf{G}_{14})$ and records $\langle \mathsf{cid} \parallel \mathsf{kid}, \mathsf{ssid}, \mathsf{cid}, [r'] \rangle$ and $\langle H_1, \mathsf{pw} \parallel \mathsf{cid}, [r] \rangle$ exist in $\text{SIM}_{\mathsf{2HDH}}$ s.t. $\bot \neq (K, \mathsf{sk}_C) \coloneqq \mathsf{AE.Dec}(y, c)$, and $b'^{1/r'} = u^{1/r}$, $(\mathbf{G}_5)$
set $i \coloneqq 0$. $(\mathbf{G}_{10})$
If more than one consistent record is found, abort the simulation. $(\mathbf{G}_4)$ // Adversarial OPRF key

b) Otherwise, if records $\langle \text{LEAKED}, \mathsf{cid}, [\mathsf{kid}'], [\mathsf{sk}_C], c, [i] \rangle$ and $\langle \text{INIT}, \mathsf{cid}, \mathsf{kid}', *, [a], *, *, *, *, *, [b], * \rangle$ exist and a record $\langle F, \mathsf{S}, *, [k], * \rangle$ exists in $\text{SIM}_{\mathsf{2HDH}}$ such that $a^k = b$ and $a'^k = b'$, $(\mathbf{G}_5)$
set $\mathsf{pw} \coloneqq \bot, K \coloneqq \bot$. $(\mathbf{G}_{10})$ // Impersonation with old file

c) Otherwise, set $\mathsf{pw} \coloneqq 0, K \coloneqq \text{FAIL}, i \coloneqq 0, \mathsf{sk}_C \coloneqq \bot$. $(\mathbf{G}_{10})$

Rb.4 If S is not honest, give input $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}^*, \mathsf{pw}, K, i)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}^*, match)$ overwrite the sixth entry in the record retrieved in Step 1 with $match$. $(\mathbf{G}_{10})$ // Ensures that we can produce output for cid

Rb.5 If HSM signature does not verify, send $(\text{COMPLETERETC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

Rb.6 If $\mathsf{cid} \neq \mathsf{cid}^*$, send $(\text{COMPLETERETC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$ // cid's a rerouted

Rb.7 If S is not fully corrupt, retrieve $\langle \text{FILE}, \mathsf{cid}, \mathsf{kid}, *, [\mathsf{sk}_C], * \rangle$. $(\mathbf{G}_{14})$

Rb.8 If $match = 1$ and $\mathsf{sk}_C \neq \bot$, compute $\sigma \coloneqq \mathsf{Sig.Sign}(\mathsf{sk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c))$, send (ssid, σ) as message from cid to S $(\mathbf{G}_{14})$
and send $(\text{COMPLETERETC}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

Rb.9 If $match = 0$ or $\mathsf{sk}_C = \bot$, $(\mathbf{G}_{14})$
send $(\text{COMPLETERETC}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{10})$

On (ssid, σ) from $\mathcal{A}$ to S on behalf of cid: // any cid, honest S

σ.1 Abort if all of the following conditions hold: $(\mathbf{G}_7)$

- There is either a record $\langle \text{FILE}, \mathsf{cid}, [\mathsf{pk}_C], *, *, * \rangle$ or SIM leaked pk_C to $\mathcal{A}$ such that $\mathsf{Sig.Vrfy}(\mathsf{pk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c), \sigma^*) = 1$, $(\mathbf{G}_7)$
- The signed c was not equivocated using $\text{SIM}_{\mathsf{EQV}}$, i.e., SIM never computed $\rho \coloneqq \text{SIM}_{\mathsf{EQV}}(c, (K, \mathsf{sk}_C))$, $(\mathbf{G}_7)$
- SIM never output σ on behalf of an honest cid. $(\mathbf{G}_7)$

σ.2 If cid is corrupt, do the following.

a) Retrieve $\langle \text{RET}, *, *, \mathsf{ssid}, *, *, \mathsf{cid}, *, [a'], [\mathsf{pk}_C], [b'], [c] \rangle$. If a record $\langle \text{DELREC}, \mathsf{ssid} \rangle$ exists, ignore. $(\mathbf{G}_1)$
Determine pw' as follows:
- If $\mathsf{Sig.Vrfy}(\mathsf{pk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c), \sigma) = 0$, set $\mathsf{pw}' = \bot$. $(\mathbf{G}_9)$
- Otherwise, search for a record $\langle H_2, [\mathsf{pw}'] \parallel *, *, [y] \rangle$ marked CONSISTENT such that $\bot \neq (K, \mathsf{sk}_C) \coloneqq \mathsf{AE.Dec}(y, c)$. If noch such a record exists, set $\mathsf{pw}' = \bot$. $(\mathbf{G}_9)$

b) Give input $(\text{RETC}, \mathsf{ssid}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PPKR}}$ on behalf of cid. On response $(\text{RETC}, \mathsf{ssid}, \mathsf{cid}, match)$ from $\mathcal{F}_{\mathsf{PPKR}}$ continue below. $(\mathbf{G}_9)$

c) Execute HSM code on input (ssid, σ) up to determining out. $(\mathbf{G}_1)$
If $out = \text{FAIL}$, send message $(\text{COMPLETERETS}, \mathsf{ssid}, 0)$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_{12})$, and otherwise $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ $(\mathbf{G}_{10})$

σ.3 If cid is honest, execute HSM code on input (ssid, σ) and
send message $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_{12})$ // If cid is honest, we only simulate σ if $match = 1$ and do not need to check again here. Due to cid-authentication the adversary also cannot inject any messages

Figure 5.15: Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ for OPRF-PPKR, part 5.

Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$, part 6.

On (ssid, σ) from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ on behalf of corrupt S: // any cid

$\sigma.12$ Abort if all of the following conditions hold: $(\mathbf{G}_7)$
- a) There is either a record $\langle \text{FILE}, \mathsf{cid}, [\mathsf{pk}_C], *, *, * \rangle$ or SIM leaked pk_C to $\mathcal{A}$ such that $\mathsf{Sig.Vrfy}(\mathsf{pk}_C, (a', \mathsf{cid}, \mathsf{ssid}, b', c), \sigma^*) = 1$, $(\mathbf{G}_7)$
- b) The signed c was not equivocated using $\text{SIM}_{\mathsf{EQV}}$, i.e., SIM never computed $\rho \coloneqq \text{SIM}_{\mathsf{EQV}}(c, (K, \mathsf{sk}_C))$, $(\mathbf{G}_7)$
- c) SIM never output σ on behalf of an honest cid. $(\mathbf{G}_7)$

$\sigma.13$ Retrieve $\langle \text{RET}, [\mathsf{cid}], *, \mathsf{ssid}, *, [match], [\mathsf{cid}^*], *, [a'], [\mathsf{pk}_C], [b'], [c] \rangle$. If a record $\langle \text{DELREC}, \mathsf{ssid} \rangle$ exists, ignore. Execute HSM code on input (ssid, σ) up to determining *out*. $(\mathbf{G}_1)$

$\sigma.14$ If $match = \bot$, give input $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}^*, \bot, \bot, \bot)$ to $\mathcal{F}_{\mathsf{PPKR}}$. On response $(\text{RETS}, \mathsf{ssid}, \mathsf{cid}^*, match)$ from $\mathcal{F}_{\mathsf{PPKR}}$ continue below. $(\mathbf{G}_{10})$ // $match = \bot$ indicates that RETS input was not given to $\mathcal{F}_{\mathsf{PPKR}}$ yet, which is necessary to decrease counter

$\sigma.15$ If $out = \text{FAIL}$, send $(\text{RETRES}, \mathsf{ssid}, \text{FAIL})$ to S. $(\mathbf{G}_1)$

$\sigma.16$ If $out = \text{SUCC}$, search for a record $\langle H_2, [\mathsf{pw}'] \parallel *, *, [y] \rangle$ marked CONSISTENT such that $\bot \neq (K, \mathsf{sk}_C) \coloneqq \mathsf{AE.Dec}(y, c)$, otherwise set $\mathsf{pw}' \coloneqq \bot$. Then do: $(\mathbf{G}_9)$ // need to reset ctr in $\mathcal{F}_{\mathsf{PPKR}}$
- a) // both messages in this retrieval come from corrupt S If $\mathsf{cid} = \bot$, give input $(\text{MALICIOUSRET}, \mathsf{cid}^*, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PPKR}}$. $(\mathbf{G}_9)$
  Independent of the response from $\mathcal{F}_{\mathsf{PPKR}}$, send $(\text{RETRES}, \mathsf{ssid}, \text{SUCC})$ to S. $(\mathbf{G}_1)$ // We give the same password as in the MALICIOUSINIT query, which ensures that the counter is reset
- b) // a' message came from cid, σ from corrupt S If $\mathsf{cid} \neq \bot$:
  - If cid is corrupt, input $(\text{RETC}, \mathsf{ssid}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PPKR}}$ $(\mathbf{G}_9)$
  - Give input $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ to $\mathcal{F}_{\mathsf{PPKR}}$ and send $(\text{RETRES}, \mathsf{ssid}, \text{SUCC})$ to S. $(\mathbf{G}_{10})$

On GETPK from anyone to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$:

pk.1 Retrieve $\langle PK, [\mathsf{pk}], * \rangle$ and return pk. $(\mathbf{G}_1)$

On LEAKFILE from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ on behalf of S:

LF.1 Send (LEAKFILE) to $\mathcal{F}_{\mathsf{PPKR}}$ to obtain L. $(\mathbf{G}_{10})$

LF.2 $L \coloneqq \emptyset$. For each (cid, ctr) in L: $(\mathbf{G}_{10})$
- a) Retrieve $\langle \text{FILE}, \mathsf{cid}, [\mathsf{kid}], [\mathsf{pk}_C], [\mathsf{sk}_C], [c] \rangle$. $(\mathbf{G}_{10})$
- b) Retrieve $\langle F, \mathsf{S}, \mathsf{cid} \parallel \mathsf{kid}, [\mathsf{k}_{\mathsf{OPRF}}], * \rangle$ in OPRF simulator. $(\mathbf{G}_{10})$
- c) If no record $\langle \text{LEAKED}, \mathsf{cid}, *, *, *, * \rangle$ exists, record $\langle \text{LEAKED}, \mathsf{cid}, \mathsf{kid}, \mathsf{sk}_C, c, 1 \rangle$. Otherwise, record $\langle \text{LEAKED}, \mathsf{cid}, \mathsf{kid}, \mathsf{sk}_C, c, i{+}1 \rangle$ where $i \in \mathbb{N}$ is the biggest number such that a record $\langle \text{LEAKED}, \mathsf{cid}, *, *, *, i \rangle$ exists. Append $(\mathsf{cid}, \mathsf{pk}_C, c, \mathsf{k}_{\mathsf{OPRF}}, ctr)$ to L. $(\mathbf{G}_{10})$

LF.3 Output L to $\mathcal{A}$. $(\mathbf{G}_{10})$

On FULLYCORRUPT from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{OPRF\text{-}PPKR}}$ on behalf of S: $(\mathbf{G}_{10})$

FC.1 Send $(\text{FULLYCORRUPT}, \mathsf{S})$ to $\mathcal{F}_{\mathsf{PPKR}}$, which outputs L. $(\mathbf{G}_{10})$

FC.2 Execute Step 2 of LEAKFILE using L. $(\mathbf{G}_{10})$

FC.3 Retrieve $\langle PK, *, [\mathsf{sk}] \rangle$ and output sk. $(\mathbf{G}_{10})$

Figure 5.16: Simulator $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ for OPRF-PPKR, part 6.

that $\mathcal{A}$ can only make a query that requires SIM to equivocate if the corresponding file containing c was leaked or if S is fully corrupt as otherwise $\mathcal{A}$ does not know the OPRF key used by $\text{SIM}_{\mathsf{2HDH}}$. This means that in $\mathcal{F}$ there is now a LEAKED record that SIM can address in the OFFLINEATTACK interface (H2.2 (a) and the first part of H2.2 (b)). This allows SIM to obtain K by using the password from the query by $\mathcal{A}$ and properly equivocate c. SIM proceeds analagously if some honest cid receives a message $(\mathsf{ssid}, b, \mathsf{cid}^*, \mathsf{pk}_{\mathsf{enc}})$ and $\mathcal{A}$ has already queried $(\mathsf{pw} \parallel \mathsf{cid}, b^{1/r})$ to H_2, except that it first has to send $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}$ in order to let $\mathcal{F}$ create the LEAKED record (Ib.6 (a)- Ib.6 (c)).

Next, we change the handling of messages $(\mathsf{ssid}, b', c, \mathsf{cid}^*)$ in SIM. Here we need to appropriately choose the values pw^*, K^*, and i for the RETS query if S is fully corrupt. For this, SIM again relies on similar checks as introduced in $\mathbf{G}_5$. If there is a record $F_{\mathsf{sid},\mathsf{S},*}(\mathsf{pw}' \parallel \mathsf{cid}^*)$ that successfully decrypts c, SIM sets $\mathsf{pw}^* := \mathsf{pw}'$ and $K^* := K$, where K is the key obtained from decrypting c (Rb.3 (a)). If no such record exists but S used the same OPRF key as in the initalization that produced c, SIM sets $\mathsf{pw}^* := \bot, K := \bot$ and chooses i such that it indicates the LEAKED record that contains c (Rb.3 (b)). Otherwise, it sets $\mathsf{pw}^* := 0$ and $K^* :=$ FAIL, which ensures that cid outputs FAIL independent of the password (Rb.3 (c)). Further, SIM uses the sk_C obtained from decrypting c, resp. the LEAKED record, to compute the signature σ.

Finally, we need to ensure that the counter for cid is updated in $\mathcal{F}$. Hence, whenever SIM gives the output $(\text{RETRES}, \mathsf{ssid}, \text{SUCC})$ to S, it additionally sends $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ to $\mathcal{F}$ (last part of σ.2 (c), σ.3 , second step of σ.16 (b)).

Let us now argue why the changes introduced in $\mathbf{G}_{10}$ are indistinguishable from $\mathbf{G}_9$. In $\mathbf{G}_9$, K was chosen uniformly at random by SIM, and K is chosen uniformly at random by $\mathcal{F}$ in $\mathbf{G}_{10}$. Therefore the distribution of K obviously does not change. Moreover, the other significant change introduced in $\mathbf{G}_{10}$ to the initialization phase is removing K from the records $\langle \mathsf{AE}, \ldots \rangle$. However, as argued above, SIM is always able to extract K from $\mathcal{F}$ whenever necessary. Thus, all outputs for any cid in the initialization phase remain unchanged.

In the retrieval phase, the removal of K from the records $\langle \mathsf{AE}, \ldots \rangle$ has no effect, since in $\mathbf{G}_9$ in the retrieval phase the record was only used to obtain K when SIM needed to output it to cid in a successful retrieval, where cid received an equivocal c. However, this is not necessary in $\mathbf{G}_{10}$, as K is output by $\mathcal{F}$ to cid.

In both phases, the ouputs of S remain unchanged as S still gets its output from SIM and the COMPLETEINITS and COMPLETERETS interfaces introduced here do not produce output to S. Overall, we therefore have

$$\Pr[\mathbf{G}_{10}] = \Pr[\mathbf{G}_9].$$

Game $\mathbf{G}_{11}$: Let $\mathcal{F}$ produce the output of honest servers in initialization. In this game, we change SIM such that it produces the output for honest servers by calling the appropriate interfaces of $\mathcal{F}$. We also change the COMPLETEINITS interface of $\mathcal{F}$ to provide output to S.

Concretely, in IC.2 (a), we still let SIM compute $(\mathsf{ssid}', \mathsf{pk}_C, c) := \mathsf{Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$ and check $\mathsf{ssid} = \mathsf{ssid}'$. However, if the check fails, SIM now sends $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ to $\mathcal{F}$ to let the server output FAIL.

Further, we remove the $(\text{COMPLETEINITS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K)$ interface from $\mathcal{F}$ that we introduced in $\mathbf{G}_8$. In $\mathbf{G}_8$ the interface was used by SIM to let $\mathcal{F}$ create FILE records. However, as of $\mathbf{G}_{10}$, $\mathcal{F}$ stores all the necessary records because of the added INITC and INITS interfaces. Therefore, SIM can execute IC.2 (d) exactly as $\text{SIM}_{\mathsf{2HDH}}$, i.e., SIM sends $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ to $\mathcal{F}$ instead of $(\text{COMPLETEINITS}, \mathsf{ssid}, \mathsf{cid}, \mathsf{pw}, K)$.

We argue that the above changes do not alter the view of $\mathcal{Z}$: First, whenever in $\mathbf{G}_{10}$ SIM would have given output $(\text{INITRES}, \mathsf{ssid}, \mathsf{cid}, \text{FAIL})$ directly to the honest S, we now use the $(\text{COMPLETEINITS}, \mathsf{ssid}, 0)$ message to $\mathcal{F}$. The effect is the same and $\mathcal{F}$ outputs $(\text{INITRES}, \mathsf{ssid}, \mathsf{cid}, \text{FAIL})$ to S. The same holds for the $(\text{COMPLETEINITS}, \mathsf{ssid}, 1)$ messages and SUCC output. In addition, COMPLETEINITS makes $\mathcal{F}$ record FILE records that use $\mathsf{cid}, \mathsf{pw}, K$ from $\mathcal{F}$'s INIT records (instead of the values provided to COMPLETEINITS). But these records were already created in $\mathbf{G}_{10}$, and therefore $\mathsf{cid}, \mathsf{pw}, K$ are the same in both games. We get

$$\Pr[\mathbf{G}_{11}] = \Pr[\mathbf{G}_{10}].$$

Game $\mathbf{G}_{12}$: Let $\mathcal{F}$ produce the output of honest servers in retrieval. In this game, we change SIM such that it produces the output for honest servers in retrieval by calling the appropriate interfaces of $\mathcal{F}$. In $\mathbf{G}_{11}$, SIM only used the COMPLETERETS interface when it wanted to produce the output SUCC for S. Hence, here we change SIM to also send the message $(\text{COMPLETERETS}, \mathsf{ssid}, 0)$ to $\mathcal{F}$ when it wants to produce the output FAIL for S (Ra.2 and σ.2 (c)). By setting $b_S = 0$, S always gets output FAIL from $\mathcal{F}$ unless $\mathcal{F}$ deletes the file for cid in this retrieval, however in that case SIM ignores the message (ssid, σ) (see σ.2 (a), σ.13). Therefore, whenver SIM sends $(\text{COMPLETERETS}, \mathsf{ssid}, 0)$ to $\mathcal{F}$, this produces the output $(\text{RETRES}\mathsf{ssid}, \text{FAIL})$ to S.

Further, it is easy to verify that any $(\text{COMPLETERETS}, \mathsf{ssid}, 1)$ query from SIM, where S is honest, produces the output $(\text{RETRES}, \mathsf{ssid}, \text{SUCC})$ to S (σ.2 (c), σ.3). In σ.2 , SIM is either able to extract the correct password due to $\mathbf{G}_9$, which leads to COMPLETERETS outputting SUCC, or if it cannot extract a password the file must have been created in a malicious initialization and COMPLETERETS then outputs SUCC as well. If cid is honest (cf. σ.3), the signature σ received in this interface is only valid if it was output by SIM, which only happens if cid used the correct password. Thus,

CompleteRetS outputs Succ as well and we have

$$\Pr[\mathbf{G}_{12}] = \Pr[\mathbf{G}_{11}].$$

Game $\mathbf{G}_{13}$: Simulate C during Init. In this game, we change how Sim computes the message (ssid, C) for honest cid as long as S is not fully corrupt. Whenever Sim receives a message $(\mathsf{ssid}, b, \mathsf{cid}, \mathsf{pk}_{\mathsf{enc}})$ to an honest cid, the simulator does not compute $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, (\mathsf{ssid}, \mathsf{pk}_C, c))$ but now computes $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, \bot)$ (see Ib.7 (b)).

We also change Sim such that it does not decrypt the ciphertext C produced for an honest cid anymore. In IC.3, when C was computed by Sim for an honest cid, then Sim retrieves the record containing pk_C and c that it stored when computing C. Similarly, in IC.6, when C was computed by Sim, then Sim just retrieves the stored values pk_C and c. Additionally, we need to check if the corrupt S replays some C that was computed as $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{enc}}, \bot)$ and if so, retrieve the corresponding values from the record instead of decrypting C (cf. IC.7 (a) and IC.7 (b)). The only change in the view of $\mathcal{Z}$ is that the distribution of the ciphertexts C. If $\mathcal{Z}$ can detect this difference, we can construct an adversary $\mathcal{B}_6$ against the $\mathsf{IND\text{-}CCA}$ security of PKE as follows:

Let $q_{\text{Init}} \in \mathbb{N}$ be the number of initializations. We construct a sequence of games $\mathbf{G}_{12}^{(0)}, \ldots, \mathbf{G}_{12}^{(q_{\text{Init}})}$, where in $\mathbf{G}_{12}^{(i)}$ the first i ciphertexts are computed as encryptions of $\bot$ and the remaining ciphertexts are encrypted as in $\mathbf{G}_{12}$. We have $\mathbf{G}_{12} = \mathbf{G}_{12}^{(0)}$ and $\mathbf{G}_{13} = \mathbf{G}_{12}^{(q_{\text{Init}})}$. Because $\mathcal{Z}$ can distinguish $\mathbf{G}_{12}$ from $\mathbf{G}_{13}$ there must be an index $i^* \in [q_{\text{Init}}]$ such that $\mathcal{Z}$ has a non-negligible advantage in distinguishing $\mathbf{G}_{12}^{(i^*-1)}$ and $\mathbf{G}_{12}^{(i^*)}$. its challenger. Now, let ssid^* denote the ssid of the i^*-th initialization and cid^* denote the cid that executes that initialization. In that initialization the reduction $\mathcal{B}_6$ does not compute $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}})$ itself in Ia.6 but uses the pk^* provided by its challenger. Then, in Ib.7 (b), $\mathcal{B}_6$ gives $m_0 \coloneqq (\mathsf{ssid}^*, \mathsf{pk}_C, c)$ and $m_1 \coloneqq \bot$ to the challenger and uses the returned C^* as ciphertext for the i^*-th initialization. In any subsequent retrieval by cid^* In IC.7 (b), when $\mathcal{B}_6$ receives a message (ssid, C) from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{HSM}}^{\mathsf{encPw}}$ on behalf of a corrupted server such that there is no record $\langle \text{Init}, *, *, \mathsf{ssid}^*, *, *, *, *, *, *, *, C \rangle$ then $\mathcal{B}_6$ uses the challenger's decryption oracle to decrypt C. Similarly, in IC.2 (a), if there is no record $\langle \text{Init}, *, *, \mathsf{ssid}^*, *, *, *, *, *, *, *, C \rangle$ then Sim again uses the challenger's decryption oracle to decrypt C.

Then we have that if C^* encrypts m_0 the game is distributed as in $\mathbf{G}_{12}^{(i^*-1)}$ and if it encrypts m_1 then the game is distributed as in $\mathbf{G}_{12}^{(i^*)}$. We get

$$|\Pr[\mathbf{G}_{13}] - \Pr[\mathbf{G}_{12}]| \leq q_{\text{Init}} \mathbf{Adv}_{\mathsf{PKE},\mathcal{B}_6}^{\mathsf{IND\text{-}CCA}}(\lambda).$$

Game $\mathbf{G}_{14}$: Remove password forwarding from $\mathcal{F}$. In this game, we do not give SIM any private input of the parties anymore. Up to $\mathbf{G}_{13}$, SIM still stored the records $\langle \mathsf{AE}, c, \mathsf{pw} \| \mathsf{cid}, b^{1/r}, \mathsf{sk}_C \rangle$ and used them for three purposes: (1) to obtain the sk_C that is supposed to be contained in an equivocable c, (2) to check whether a client used the correct password in a retrieval, and (3) to determine critical queries $(\mathsf{pw} \parallel \mathsf{cid}, b^{1/r})$ to H_2 that require SIM to equivocate c. As we now finally remove the AE records, we have to simulate these steps in another way.

Issue (1) can be solved trivially, and we instead use the FILE records, which also store c and sk_C (see Rb.7). For issue (2), we now rely on the bit *match* that SIM receives from $\mathcal{F}$ and indicates whether the password used in the retrieval is correct (see Rb.8 and Rb.9).

To solve issue (3), we can instead rely on the records $\langle H_2, \mathsf{pw} \parallel \mathsf{cid}, u, y \rangle$ and the CONSISTENT marking. An H_2 record if marked CONSISTENT if and only if $u = H_1(\mathsf{pw} \parallel \mathsf{cid})^k$ for some OPRF key k, where k can be either created by $\text{SIM}_{\mathsf{2HDH}}$ (cf. H2.2), i.e., when S is not fully corrupt, or adversarially chosen (cf. Ib.6 (b)), i.e., when S is fully corrupt. Then, whenever SIM would query the OFFLINEATTACK interface in $\mathbf{G}_{13}$ with the password pw obtained from the AE record, in this game we instead query the interface with all passwords pw from CONSISTENT H_2 records (Ib.6 (b)).

With this change, we finally reach the point where $\text{SIM} = \text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$ and $\mathcal{F} = \mathcal{F}_{\mathsf{PPKR}}$. We clearly added all interfaces of $\mathcal{F}_{\mathsf{PPKR}}$ to $\mathcal{F}$. One can also verify that SIM is indeed $\text{SIM}_{\mathsf{OPRF\text{-}PPKR}}$.

5.5 Evaluation & Discussion

In this section, we give an overview of the concrete efficiency of our protocols, compare them to WBP, and also discuss the respective advantages of our protocols.

Instantiations of Building Blocks For the efficiency overview, we choose concrete instantiations of the required primitives, such that we can count the number of operations performed in each protocol. We only chose group-based public key primitives to keep the numbers comparable. But of course, one could use any other secure instantiation of the primitives, e.g., based on RSA or lattices.

Concretely, we chose Schnorr-Signatures, HMAC, HKDF, ElGamal encryption as a CPA secure encryption and DHIES as CCA secure encryption. For simplicity, we assumed that all hash evaluations cost a uniform unit "1 Hash"[3] and similarly that one AE encryption or decryption costs "1 AES". A detailed overview of the computation costs of each primitive can be found in Table 5.2. We did not list the HSM's attestation signatures as they are automatically produced by the HSM anyway.

[3]Ignoring e.g., exponentiations that might be needed to hash into a group.

Table 5.2: Costs of concrete building blocks for the efficiency evaluation in Section 5.5.

	KGen	Enc	Dec
CPA Enc (ElGamal)	1 Exp	2 Exp, 1 Hash	1 Exp, 1 Hash
CCA Enc (DHIES)	1 Exp	2 Exp, 3 Hash, 1 AES	1 Exp, 3 Hash, 1 AES
AE (AES-GCM)	-	1 AES	1 AES

	KGen	Sign	Vrfy
Signature (Schnorr)	1 Exp	1 Exp,1 Hash	2 Exp, 1 Mult, 1 Hash
MAC (HMAC)	-	2 Hash	2 Hash

KDF (n keys, HKDF)	$(2n + 2)$ Hash

Table 5.3: Efficiency of PPKR realizations expressed in terms of the number of exponentiations (E), multiplications (M), hash evaluations (H) and AES encryptions/decryptions (A). Since the server mostly just relays messages, we ignore its costs in the comparison. In case of encPw and WBP, we assume that the static encryption key of the HSM is hardcoded into the client, therefore no communication is needed for key sharing in Init and Rec and we neglect the costs of the one-time generation. For more details on the instantiations of the primitives used for the comparison, we refer to Table 5.2.

		encPw **(Sec. 5.2)**	encPw+ **(Sec. 5.3)**	OPRF-PPKR **(Sec. 5.4)**	WBP **(Sec. 4.2)**
Init	Client	2 E, 3 H, 1 A	2 E, 3 H, 1 A	5 E, 5 H, 2 A	7 E, 12 H, 1 A, 1 M
	HSM	3 H, 1 A	2 E, 5 H, 1 A	2 E, 3 H, 1 A	3 E, 3 H
	no. rounds	2	3	3	3
Rec	Client	2 E, 3 H, 2 A	2 E, 3 H, 2 A	3 E, 3 H, 1 A	8 E, 27 H, 2 A, 1 M
	HSM	3 H, 2 A	2 E, 5 H, 2 A	2 E, 1 H, 1 M	6 E, 15 H, 1 A
	no. rounds	2	3	3	4

Efficiency Comparison As Table 5.3 shows, OPRF-PPKR is more efficient than the WBP in both, initialization and retrieval. This comes mostly from WBP performing an authenticated key exchange where OPRF-PPKR uses a digital signature. In particular, in the retrieval phase, which will be the more time-critical phase in deployment, as it will be run more often than initialization, our protocol outperforms WBP: OPRF-PPKR reduces the round[4] complexity from 4 to 3, and uses roughly one-third of the operations required by WBP, for both the client and HSM.

If we compare the two enhanced encrypt-to-HSM protocols to OPRF-PPKR , they are more efficient and they save one round of communication, as they strongly rely on the HSM security, with the weakest protocol having the lowest computa-

[4]We count one round as a message from party A to party B (and not as a full round-trip A to B to A).

tional requirements. Interestingly, in bare numbers, they are not significantly more efficient though.

Thus, considering that OPRF-PPKR provides much better security for almost the same costs, this might raise the question of whether there are any advantages in using our simpler protocols – which is what we answer next.

Advantages of Standard Primitives The core benefit of our two basic/enhanced encrypt-to-HSM protocols is that they explore how the extended trust in the HSM can be traded for simplicity in the protocol design. Both protocols rely on standard and well-understood primitives only, which are public-key and symmetric encryption, and hash functions. In contrast, WBP and our OPRF-PPKR require (dedicated discrete-log-based) OPRFs.

In particular, when internally using an HSM, reliance on simple and standard building blocks is an advantage – established primitives are implemented in many well-tested frameworks and have been studied for resistance against side-channel attacks. OPRFs are still a somewhat more modern primitive that just recently came into focus of practitioners, i.e., it might require developers to implement low-level cryptographic procedures instead of merely invoking APIs of trusted libraries. Thus, the operations needed for OPRF-PPKR and WBP might be more prone to implementation errors or be simply not available or accessible through the shielded HSM APIs when using "off-the-shelf" HSMs. The clear downside of both simple protocols is that they lose security when the HSM is (fully) compromised. However, given that they require only well-understood operations by the HSM, such a simpler HSM might be easier to protect, making a security breach less likely.

Further, realizing an efficient *quantum-safe* OPRF is still an open problem. This is in contrast to the other standard building blocks for which quantum-safe options exist. Here the basic/enhanced encrypt-to-HSM protocols again have the advantage over OPRF-PPKR and WBP, as they rely on standard primitives only, and can easily benefit from a post-quantum "upgrade".

Lev-3 Security for Offline Users in enhanced encrypt-to-HSM While we prove the enhanced encrypt-to-HSM protocol to satisfy at most Lev-2 security, it actually does preserve strong guarantees if the HSM is fully corrupted, but only for "offline" clients. Recall that we assumed the HSM's attestation key to be the single value that enjoys particularly strong protection – and thus is the only additional information the adversary gets upon full server corruption. Consequently, the impact of such a compromise on the protocol's security is then rather limited. In fact, for users who never use the PPKR *after* the full corruption happened, our enhanced encrypt-to-HSM protocol provides the same protection as OPRF-PPKR : their leaked files must still be cracked through individual offline attacks against each password. This might be a sufficient guarantee in reality. Especially in settings where it will be known which HSM is used by the server, and assuming that

a breach of the HSM most critical operation would become public. Then either the service is stopped, and the server updates to a secure HSM; or the particularly cautious users would no longer log into the PPKR service – and their password and key would be just as secure as with the OPRF-PPKR protocol. Only the users who still engage in active sessions would lose their security. However, their security is fully compromised in enhanced encrypt-to-HSM, as the malicious server will learn their password and key in plain as soon as they start a new initialization or retrieval session. Here, OPRF-PPKR still ensures the secrecy of the user's data.

What is the best protocol? Overall, there is no clear answer to what can be considered the "best" protocol. We believe that both our enhanced encrypt-to-HSM and OPRF-PPKR protocols have their individual strengths that can make each the right choice for a dedicated deployment setting. The enhanced encrypt-to-HSM is clearly superior to its unsalted variant, yet preserves all simplicity advantages. Thus, here we do not see a strong reason to favor the encPw protocol (Lev-1 security) and would recommend opting for the encPw+ version (Lev-2 security) whenever the advantages of the simple and standard construction outweigh the concerns of a full HSM corruption. As just discussed, the guarantees of encPw+ in case of full corruption are actually not lost entirely. For those users who never use the retrieval again after the full HSM corruption occurred, it provides the same security as OPRF-PPKR. Nevertheless, for applications with very high-security requirements, the OPRF-PPKR construction provides the strongest (Lev-3) guarantees, but requires more care in the implementation.

6 Conclusion

Let us again summarize our contributions in this Part. In Chapter 4, we have presented the first formal security analysis of the widely-used WhatsApp backup protocol and confirmed that the WBP indeed provides strong security guarantees such as online protection of the password, and the strength and secrecy of the backup key. However, we also show how a compromised WhatsApp server can increase the number of admissible password guesses from only ctr on the *most recent* password of the user, to $q_{\mathsf{pw}} * \mathsf{ctr}$ on *any* password pw ever entered by the user, where q_{pw} is the number of initializations performed with pw by the WhatsApp client device.

In Chapter 5, we proposed the three PPKR protocols $\mathsf{encPw}, \mathsf{encPw}+$, and $\mathsf{OPRF\text{-}PPKR}$. The protocols provide a tradeoff which trust assumptions one is willing to accept. On one end of the spectrum, if we want the highest security guarantees, the OPRF-PPKR protocol provides strong security guarantees even if the HSM looses all its guarantees. Essentially, an adversary in control of the HSM can still not do any better than trying to guess passwords in that case. On the other end of the spectrum, lies the basic encrypt-to-HSM protocol, which offers great performance, and as long the HSM does not leak any data, the protocol provides the same security guarantees as $\mathsf{OPRF\text{-}PPKR}$.

6.1 Future Research Directions

Let us discuss some possible future research directions for PPKR.

Further analysis of the WBP. Having we already defined the security model for corruption of the HSM, a natural question is whether the security guarantees of the WBP are retained in that scenario as well. It seems reasonable to conjecture that they hold thank to the WBP being build on top of OPAQUE. Nevertheless, even in the case where the HSM is perfectly secure, the WBP has some subtle issues that we had to account for in the security analysis by artificially weakening $\mathcal{F}_{\mathsf{PPKR}}$. For this reason, a thorough analysis of Lev-2 and Lev-3 security is necessary before any claims can be made for the WBP in these scenarios.

Another possible direction is the analysis of the outsourced storage protocol that WhatsApp employs to avoid storing extensive user data directly on the HSM. This could take two approaches. (1) Which security guarantees does the solution deployed by WhatsApp provide, or (2) which security guarantees to we require

from an outsourced storage protocol such that it can be used as a building block for PPKR, possibly under the different corruption scenarios.

New constructions of PPKR. As already discussed in Section 5.5, there is an advantage in using only standard building blocks to construct PPKR. Hence, it is an interesting question, whether we can develop a PPKR scheme with Lev-3 security from only standard building blocks. Another potential research direction would be exploring which level of security we can achieve without the help of some trusted hardware.

Bibliography

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158, San Francisco, CA, USA, April 8–12, 2001. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-45353-9_12.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-17653-2_5.

[Bad14] Christoph Bader. Efficient signatures with tight real world security in the random-oracle model. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14*, volume 8813 of *LNCS*, pages 370–383, Heraklion, Crete, Greece, October 22–24, 2014. Springer, Cham, Switzerland. doi:10.1007/978-3-319-12280-9_24.

[BC19] Jean-Baptiste Bedrune and Gabriel Campana. Everybody be cool, this is a robbery! BlackHat USA 2019, 2019. http://i.blackhat.com/USA-19/Thursday/us-19-Campana-Everybody-Be-Cool-This-Is-A-Robbery.pdf, Accessed: 24.04.2024.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_3.

[BEL+20] Julian Brost, Christoph Egger, Russell W. F. Lai, Fritz Schmid, Dominique Schröder, and Markus Zoppelt. Threshold password-hardened encryption services. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 409–424, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3417266.

[BFG+22] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the Signal double ratchet algorithm. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 784–813, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-15802-5_27.

[BG11] Colin Boyd and Juan Manuel González Nieto. On forward secrecy in one-round key exchange. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *LNCS*, pages 451–468, Oxford, UK, December 12–15, 2011. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-25516-8_27.

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658, Warsaw, Poland, March 23–25, 2015. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_26.

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304, Vienna, Austria, May 8–12, 2016. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_10.

[BJSL11] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 433–444, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. doi:10.1145/2046707.2046758.

[BKLW22] Daniel Bourdrez, Dr. Hugo Krawczyk, Kevin Lewi, and Christopher A. Wood. The OPAQUE Asymmetric PAKE Protocol. Internet-Draft draft-irtf-cfrg-opaque-09, Internet Engineering Task Force, July 2022. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/09/.

[BMS20] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for Authentication and Key Establishment, Second Edition*. Information Security and Cryptography. Springer, 2020. doi:10.1007/978-3-662-58146-9.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-45539-6_11.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-48329-2_21.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66, Las Vegas, NV, USA, May 29 – June 1, 1995. ACM Press. doi:10.1145/225058.225084.

[BR09] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 407–424, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-01001-9_24.

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-63697-9_21.

[BWJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *LNCS*, pages 30–45, Cirencester, UK, December 17–19, 1997. Springer, Berlin, Heidelberg, Germany. doi:10.1007/bfb0024447.

[BWM99] Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols (invited talk). In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 339–361, Kingston, Ontario, Canada, August 17–18, 1999. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-48892-8_26.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. doi:10.1109/SFCS.2001.959888.

[Cat22] Will Cathcart. https://twitter.com/wcathcart/status/1600603826477617152, December 2022.

[CCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *EuroS&P*, pages 451–466. IEEE, 2017.

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26954-8_25.

[CF15] Cas Cremers and Michèle Feltz. Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Des. Codes Cryptogr.*, 74(1):183–218, 2015. URL: https://doi.org/10.1007/s10623-013-9852-1.

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. In David Evans and Carmela Troncoso, editors, *IEEE EuroS&P 2022*. IEEE, 2022.

[CJSV22] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. Universally composable end-to-end secure messaging. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 3–33, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-15979-4_1.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474,

Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-44987-6_28.

[CLN12] Jan Camenisch, Anna Lysyanskaya, and Gregory Neven. Practical yet universally composable two-server password-authenticated secret sharing. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 525–536, Raleigh, NC, USA, October 16–18, 2012. ACM Press. doi:10.1145/2382196.2382252.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-46035-7_18.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3417887.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA. doi:10.1007/0-387-34805-0_39.

[DFG+23] Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. Security analysis of the WhatsApp end-to-end encrypted backup protocol. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 330–361, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38551-3_11.

[DFHSW23] Alex Davidson, Armando Faz-Hernandez, Nick Sullivan, and Christopher A. Wood. Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups. Internet-Draft draft-irtf-cfrg-voprf-17, Internet Engineering Task Force, January 2023. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-voprf/17/.

[DFW19] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. Cryptology ePrint Archive, Report 2019/1203, 2019. https://eprint.iacr.org/2019/1203.

[DFW20] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 288–303, Boston, MA, USA, June 22–26, 2020. IEEE Computer Society Press. doi:10.1109/CSF49147.2020.00028.

[DG21] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21International Conference on Applied Cryptography and Network Security, Part II*, volume 12727 of *LNCS*, pages 448–479, Kamakura, Japan, June 21–24, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-78375-4_18.

[DGJL21] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. More efficient digital signatures with tight multi-user security. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 1–31, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-75248-4_1.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DHL22] Poulami Das, Julia Hesse, and Anja Lehmann. DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 682–696, Nagasaki, Japan, May 30 – June 3, 2022. ACM Press. doi:10.1145/3488932.3517389.

[Die23] Denis Diemert. *On the tight security of the Transport Layer Security (TLS) Protocol Version 1.3*. PhD thesis, Veröffentlichungen der Universität, Wuppertal, 2023. Publication Title: Fakultät für Elektrotechnik, Informationstechnik und Medientechnik. URL: https://nbn-resolving.org/urn:nbn:de:hbz:468-2-983, doi:10.25926/BUW/0-98.

[DJ21] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021. doi:10.1007/s00145-021-09388-x.

[DLS21] Gérald Doussot, Marie-Sarah Lacharité, and Eric Schorn. End-to-End Encrypted Backups Security Assessment. https://research.nccgroup.com/wp-content/uploads/2021/10/

NCC_Group_WhatsApp_E001000M_Report_2021-10-27_v1.2.pdf, October 2021.

[DvW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *DCC*, 2(2):107–125, 1992. doi:10.1007/BF00124891.

[EH11] D. Eastlake and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, IETF, May 2011. URL: http://tools.ietf.org/rfc/rfc6234.txt.

[EJ01] D. Eastlake and P. Jones. US Secure Hash Algorithm 1. RFC 3174, IETF, September 2001. URL: http://tools.ietf.org/rfc/rfc3174.txt.

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 444–460, Athens, Greece, May 26–30, 2013. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_27.

[FGSW16] 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*, pages 452–469, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press. doi:10.1109/SP.2016.34.

[FHH+24] Sebastian H. Faller, Tobias Handirk, Julia Hesse, Máté Horváth, and Anja Lehmann. Password-protected key retrieval with(out) HSM protection. In *ACM CCS 2024*, Salt Lake City, Utah, USA, October 14–18, 2024. ACM Press. To appear.

[FJS19] Nils Fleischhacker, Tibor Jager, and Dominique Schröder. On tight security proofs for Schnorr signatures. *Journal of Cryptology*, 32(2):566–599, April 2019. doi:10.1007/s00145-019-09311-5.

[GGJJ23] Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. On optimal tightness for key exchange with full forward secrecy via key confirmation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 297–329, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38551-3_10.

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume

10992 of *LNCS*, pages 95–125, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96881-0_4.

[GJK21] Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. KHAPE: Asymmetric PAKE from key-hiding key exchange. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 701–730, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-84259-8_24.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMR06] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Berlin, Heidelberg, Germany. doi:10.1007/11818175_9.

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37, Houthalen, Belgium, April 10–13, 1990. Springer, Berlin, Heidelberg, Germany. doi:10.1007/3-540-46885-4_5.

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 66–83, Darmstadt, Germany, May 21–23, 2012. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-30057-8_5.

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden,

and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 389–422, Edinburgh, UK, May 4–7, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45388-6_14.

[HLWG23] Shuai Han, Shengli Liu, Zhedong Wang, and Dawu Gu. Almost tight multi-user security under adaptive corruptions from LWE in the standard model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 682–715, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38554-4_22.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-662-45608-8_13.

[JKKX16] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 276–291. IEEE, 2016.

[JKR19] Stanislaw Jarecki, Hugo Krawczyk, and Jason K. Resch. Updatable oblivious key management for storage systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 379–393, London, UK, November 11–15, 2019. ACM Press. doi:10.1145/3319535.3363196.

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-77870-5_5.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-78372-7_15.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. In Dennis Hofheinz and Alon Rosen,

editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 180–210, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-36033-7_7.

[Kev23] Direct correspondences with Kevin Lewi and other members of the WhatsApp engineering team, 2022-2023.

[KL21] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall, CRC Press, third edition, 2021.

[Kle08] John C. Klensin. Simple Mail Transfer Protocol. RFC 5321, IETF, October 2008. URL: http://tools.ietf.org/rfc/rfc5321.txt.

[KLW21] Dr. Hugo Krawczyk, Kevin Lewi, and Christopher A. Wood. The OPAQUE Asymmetric PAKE Protocol. Internet-Draft draft-irtf-cfrg-opaque-03, Internet Engineering Task Force, February 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/03/.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_24.

[Kra96] Hugo Krawczyk. SKEME: a versatile secure key exchange mechanism for internet. In James T. Ellis, B. Clifford Neuman, and David M. Balenson, editors, *NDSS'96*, pages 114–127, San Diego, CA, USA, February 22–23, 1996. IEEE Computer Society. doi:10.1109/NDSS.1996.492418.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Heidelberg, Germany. doi:10.1007/11535218_33.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_34.

[Krs16] Ivan Krstic. Behind the scenes with ios security, 2016. https://www.blackhat.com/docs/us-16/materials/us-16-Krstic.pdf, Accessed: 18.04.2024.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LER+18] Russell W. F. Lai, Christoph Egger, Manuel Reinert, Sherman S. M. Chow, Matteo Maffei, and Dominique Schröder. Simple password-hardened encryption services. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1405–1421, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 785–814, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-64834-3_27.

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-540-75670-5_1.

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1343–1360, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3134006.

[Lun19] Joshua Lund. Technology preview for secure value recovery, 2019. https://signal.org/blog/secure-value-recovery/, Accessed: 18.04.2024.

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_4.

[Mau11] Ueli Maurer. Constructive cryptography–a new paradigm for security definitions and proofs. In *Joint Workshop on Theory of Security and Applications*, pages 33–56. Springer, 2011.

[ML21] Alexey Melnikov and Barry Leiba. Internet Message Access Protocol (IMAP) - Version 4rev2. RFC 9051, IETF, August 2021. URL: http://tools.ietf.org/rfc/rfc9051.txt.

[Nov18] Matt Novak. Paul Manafort Learns That Encrypting Messages Doesn't Matter If the Feds Have a Warrant to Search Your iCloud Account. https://gizmodo.com/paul-manafort-learns-that-encrypting-messages-doesnt-ma-1826561511, June 2018.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.

[Orm98] Hilarie Orman. The OAKLEY Key Determination Protocol. RFC 2412, IETF, November 1998. URL: http://tools.ietf.org/rfc/rfc2412.txt.

[OSV23] Chris Orsini, Alessandra Scafuro, and Tanner Verber. How to recover a cryptographic secret from the cloud. Cryptology ePrint Archive, Paper 2023/1308, 2023. https://eprint.iacr.org/2023/1308. URL: https://eprint.iacr.org/2023/1308.

[Per] Trevor Perrin. The noise protocol framework. URL: http://noiseprotocol.org/noise.html.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PRZ24] Jiaxin Pan, Doreen Riepel, and Runzhi Zeng. Key exchange with tight (full) forward secrecy via key confirmation. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 59–89, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-58754-2_3.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20, Chennai, India, December 4–8, 2005. Springer, Berlin, Heidelberg, Germany. doi:10.1007/11593447_1.

[PW22] Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In Goichiro Hanaoka, Junji

Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 347–378, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-030-97131-1_12.

[PWZ23] Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Tighter security for generic authenticated key exchange in the QROM. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 401–433, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore. doi:10.1007/978-981-99-8730-6_13.

[Res18] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, August 2018. URL: http://tools.ietf.org/rfc/rfc8446.txt.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, IETF, April 1992. URL: http://tools.ietf.org/rfc/rfc1321.txt.

[RMS18] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *EuroS&P*, pages 415–429. IEEE, 2018.

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 211–228, Hanoi, Vietnam, September 25–28, 2006. Springer, Berlin, Heidelberg, Germany. doi:10.1007/11958239_14.

[RZ18] Phillip Rogaway and Yusi Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 3–32, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96881-0_1.

[Sca19] Alessandra Scafuro. Break-glass encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 34–62, Beijing, China, April 14–17, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-17259-6_2.

[Sho99] Victor Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. https://eprint.iacr.org/1999/012.

[SRW22] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust dies in darkness: Shedding light on samsung's TrustZone

keymaster design. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 251–268, Boston, MA, August 2022. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/shakevsky.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1461–1480, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3423350.

[TB22] Martin Thomson and Cory Benfield. HTTP/2. RFC 9113, IETF, June 2022. URL: http://tools.ietf.org/rfc/rfc9113.txt.

[Tsu92] Gene Tsudik. Message authentication with one-way hash functions. *ACM SIGCOMM Computer Communication Review*, 22(5):29–38, 1992.

[VBMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[VBPS17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Sgx-step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, pages 1–6, 2017.

[VGIK20] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. On the cryptographic deniability of the Signal protocol. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20International Conference on Applied Cryptography and Network Security, Part II*, volume 12147 of *LNCS*, pages 188–209, Rome, Italy, October 19–22, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-57878-7_10.

[Wal18] Shabsi Walfish. Google cloud key vault service, 2018. https://developer.android.com/about/versions/pie/security/ckv-whitepaper, Accessed: 18.04.2024.

[Wha21a] WhatsApp. Security of End-to-End Encrypted Backups. https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf, September 2021.

[Wha21b] WhatsApp. WhatsApp Encryption Overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, November 2021.

[Yan13a] Zheng Yang. Efficient eCK-secure authenticated key exchange protocols in the standard model. In Sihan Qing, Jianying Zhou, and Dongmei Liu, editors, *ICICS 13*, volume 8233 of *LNCS*, pages 185–193, Beijing, China, November 20–22, 2013. Springer, Cham, Switzerland. doi:10.1007/978-3-319-02726-5_14.

[Yan13b] Zheng Yang. Modelling simultaneous mutual authentication for authenticated key exchange. In *FPS*, volume 8352 of *Lecture Notes in Computer Science*, pages 46–62. Springer, 2013.

A Comparison of the WBP to the OPAQUE Internet Draft Notation

In Table A.1 we list naming differences between our description of the WBP and the notation used in the OPAQUE draft. In the following we summarize where our protocol description in the Figures 4.4 and 4.5 differs from the OPAQUE draft [KLW21], and justify these changes.

- In the OPAQUE draft the server sends its public key in its first message in both phases (see Sections 3.3.2 and 4.1.2.2 in [KLW21]). This public key is already hardcoded into the WhatsApp client and thus is not sent by the HSM.
- In the OPAQUE draft $K_{\mathsf{export}}, K_{\mathsf{mask}}, K_{\mathsf{auth}}$ are derived via a memory-hard function and HKDF [Kra10] (see Section 3.3.3 in [KLW21]). These steps are simplified into one computation via KDF_1 in Figures 4.4 and 4.5.
- According to the OPAQUE draft, pre contains either the identities of the client and the server, or their respective public keys (see Section 6.2 in [KLW21]). In the WBP, pre contains only the public key of the HSM and neither the client's identity nor its public key. However, pre contains $\mathsf{e\_cred}$, which is an encryption of sk_C. Since T_e ensures the integrity and authenticity of $\mathsf{e\_cred}$, $\mathsf{e\_cred}$ uniquely determines the client's public key, which means that it still is implicitly contained in pre.
- In the OPAQUE draft $\mathsf{K}_S^{\mathsf{MAC}}, \mathsf{K}_C^{\mathsf{MAC}}, \mathsf{shk}$ are derived via a series of HKDF computations (see Section 4.2.2.2 in [KLW21]). These steps are simplified into one computation via KDF_2 in Figure 4.5.

Table A.1: Names and variables as they are referred to in our protocol description and in the OPAQUE Internet Draft [KLW21].

Our notation	**OPAQUE Internet Draft notation**
S	`Server`
cid	`Client`
K_{export}	`export_key`
r_1 and r_2	`blind`
a_1 and a_2	`request` and `M`
$(\mathsf{pk}_C, \mathsf{sk}_C)$	`creds`
(y, Y)	`(server_private_key, server_public_key)`
y	`secret_creds`
Y	`cleartext_creds`
pw	`password`
$\mathsf{k}_{\mathsf{OPRF}}$	`oprf_key`
n_e	`envelope_nonce`
K_{mask}	`pseudorandom_pad`
K_{auth}	`auth_key`
$\mathsf{e\_cred}$	`encrypted_creds`
T_e	`auth_tag`
$(n_e, \mathsf{e\_cred}, T_e)$	`envelope`
$(\mathsf{k}_{\mathsf{OPRF}}, \mathsf{pk}_C, (n_e, \mathsf{e\_cred}, T_e))$	`credential_file`
T'_e	`expected_tag`
n_C	`client_nonce`
U	`client_keyshare` and `epkU`
V	`server_keyshare` and `epkS`
u	`eskU`
v	`eskS`
n_S	`server_nonce`
T_S or T'_C	`mac`
$\mathsf{K}_S^{\mathsf{MAC}}$	`server_mac_key`
$\mathsf{K}_C^{\mathsf{MAC}}$	`client_mac_key`
ikm	`IKM`
shk	`session_key`

B On not Using Proven OPAQUE Guarantees for the WBP

In this section we expand on the reasoning why the proven security guarantees of OPAQUE [JKX18] do not carry over directly to the WBP, in particular the differences between the OPAQUE version proven secure therein and the version from Internet Draft v03 [KLW21] that the WBP is using.

In order to modularly use an saPAKE functionality, we would need to formally prove which exact functionality Internet Draft v03 [KLW21] (or, more specific, the OPAQUE protocol as implemented in Figures 4.4 and 4.5) UC-emulates. This however seems overkill, because a) the list of differences is quite extensive, and b) we do not even rely on the "full" OPAQUE security: we only rely on OPAQUE being secure against a malicious client, because the OPAQUE server code is run on an incorruptible HSM.

For completeness, we list below the differences between the proven OPAQUE protocol and the one deployed in WBP.

- OPAQUE from Internet Draft v03 [KLW21] does not separate hash domains of the different 2HashDH PRFs with domain separators. [JKX18] only analyzes the security of OPAQUE where the hash domains are separate on a per-PRF-key basis.
- OPAQUE from Internet Draft v03 [KLW21] has a long-term public key pair of the server (`server_private_key`, `server_public_key`), which is used for the generation of every password file. The paper version of OPAQUE [JKX18, Fig. 8] lets the server choose a fresh key pair (p_S, P_S) for every password file.
- OPAQUE from Internet Draft v03 [KLW21] has an interactive registration phase where the server does not learn the password. The paper version of OPAQUE [JKX18, Fig. 8] has a registration phase where the server gets the cleartext password as input. This difference was already pointed out in [BKLW22], where it is claimed that (1) interactive registration only adds to the security proven in [JKX18], and (2) an upcoming paper analyzes the interactive phase.
- OPAQUE from Internet Draft v03 [KLW21] puts the client secret key sk_C in the password file in form of $\mathsf{e\_cred} := \mathsf{sk}_C \oplus K_{\mathsf{mask}}$. In the paper version of OPAQUE [JKX18, Fig. 8], a password file contains password-encrypted credentials $\mathsf{AuthEnc}_{rw}(p_u, P_u, P_S)$, where rw is the PRF value for pw. A user can retrieve their key pair p_u, P_u from the file by decrypting it with rw. Draft v09 [BKLW22] motivates this change by (1) smaller password files and (2) no

need for applications to provide AKE key material to the client, but instead deal with key generation within OPAQUE. (2) is actually only a difference between v09 and older versions of the draft, because in [JKX18] OPAQUE (more specific: the server) is generating the client's AKE key pairs. Draft v09 [BKLW22] also says that this change is analyzed in an upcoming paper.

- OPAQUE from Internet Draft v03 [KLW21] lets the client output an additional export key `export_key` that is not present in [JKX18].
- OPAQUE from Internet Draft v03 [KLW21] uses 3DH as AKE while [JKX18] only shows that HMQV can be used.
- OPAQUE from Internet Draft v03 [KLW21] uses a superset of transcripts used in [JKX18].

C Explicit entity authentication vs. explicit key authentication.

Here we prove that explicit *entity* authentication reduces to explicit *key* authentication. This is basically a restatement of Proposition 9 in [DFW19][1]. Note that Proposition 8 describes a reduction for the *same* protocol Π unlike in Section 3.4 where the reductions are from the extended protocol Π^+ to the underlying protocol Π.

Proposition 8. Let Π be a protocol and $\mathcal{A}$ and adversary against Π for (almost-)full entity authentication. Then $\mathcal{A}$ is also an adversary against match soundness, implicit key authentication, key match soundness and (almost-)full key authentication, and

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A})$$

$$\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A})$$

Proof. The proofs of full and almost-full explicit entity authentication are identical so we only give a proof of the former.

Suppose $s \in \mathcal{L}_{\mathsf{rcv}}$ is a session that receives the last message of the protocol, and for which $\mathsf{fexEntAuth}$ is violated (so s has accepted). We consider two cases:

1. There is a session s' such that $\mathsf{Partner}(s, s') = \mathsf{true}$, but $s.\mathsf{peer} \neq s.\mathsf{party}$. We consider two further sub-cases:
   a) $\mathsf{SameKey}(s, s') \neq \mathsf{true}$. In this case Match is violated.
   b) $\mathsf{SameKey}(s, s') = \mathsf{true}$. In this case $\mathsf{iKeyAuth}$ is violated.
2. There is no session s' such that $\mathsf{Partner}(s, s') = \mathsf{true}$, but $\mathsf{aFresh}(s) = \mathsf{true}$. We consider two further sub-cases:
   a) There is session s'' such that $\mathsf{SameKey}(s, s'') = \mathsf{true}$. In this case $\mathsf{KMSound}$ is violated.
   b) There is no session s' such that $\mathsf{SameKey}(s, s') = \mathsf{true}$. In this case $\mathsf{fexKeyAuth}$ is violated.

□

[1] Although [DFW19, Prop. 9] appears to miss an $\mathsf{iKeyAuth}$ term.