UC Irvine
UC Irvine Electronic Theses and Dissertations

Title

New Paradigms For Efficient Password Authentication Protocols

Permalink

https://escholarship.org/uc/item/7qm0220s

Author

Gu, Yanqi

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

New Paradigms For Efficient Password Authentication Protocols

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Yanqi Gu

Dissertation Committee:
Professor Stanislaw Jarecki, Chair
Professor Athina Markopoulou
Professor Michael Goodrich

2024

Chapter 3 © 2021 Springer, Cham
Chapter 4 © 2022 Springer, Cham
Chapter 5 © 2023 Springer, Cham
All other materials © 2024 Yanqi Gu

DEDICATION

To my parents

TABLE OF CONTENTS

	Page
LIST OF FIGURES	**v**
LIST OF TABLES	**viii**
ACKNOWLEDGMENTS	**ix**
VITA	**x**
ABSTRACT OF THE DISSERTATION	**xiii**
1 Introduction	**1**
1.1 Problem Statement	1
1.1.1 Roadmap	4
2 Preliminaries	**5**
2.1 Cryptographic Assumptions	5
2.2 Cryptographic Primitives	5
2.3 Security Models and Frameworks	9
2.3.1 The Random Oracle Model and Ideal Cipher Model	9
2.3.2 Universally Composability Framework	9
2.3.3 Functionalities	10
3 KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange	**17**
3.1 Introduction	17
3.2 The Key-Hiding AKE UC Functionality	21
3.3 3DH as Key-Hiding AKE	29
3.4 HMQV as Key-Hiding AKE	43
3.5 SKEME as Key-Hiding AKE	53
3.6 Compiler from key-hiding AKE to aPAKE	63
3.7 Concrete aPAKE Instantiation: KHAPE-HMQV	80
3.8 Curve Encodings and Ideal Cipher	82
3.8.1 Quasi bijections	82
3.8.2 Implementing quasi-bijective encodings	83
3.8.3 Ideal Cipher Constructions	85

4 OKAPE:Asymmetric PAKE with low computation and communication 87
4.1 Introduction 87
4.2 Key-hiding one-time-key AKE 94
4.2.1 2DH as key-hiding one-time-key AKE 98
4.2.2 One-Pass HMQV as key-hiding one-time-key AKE 110
4.2.3 1/2-SKEME as one-time-key AKE 119
4.3 Protocol OKAPE: asymmetric PAKE construction 130

5 Randomized Half-Ideal Cipher on Groups with application to UC (a)PAKE 152
5.1 Introduction 152
5.2 Universally Composable Randomized Ideal Cipher 162
5.3 Randomized Ideal Cipher Construction: Modified 2-Feistel 165
5.4 Encrypted Key Exchange with Randomized Ideal Cipher 183
5.4.1 EKE with Randomized Ideal Cipher : the KEM version 197
5.5 Applications of HIC to asymmetric PAKE 207
5.6 Lattice-Based UC PAKE from EKE and Saber KEM 219

6 Generic compiler from PAKE to asymmetric PAKE using KEM 224
6.1 Introduction 224
6.1.1 Prior aPAKE Constructions 226
6.1.2 Our Contributions 232
6.2 Compiler from PAKE to asymmetric PAKE 237
6.3 An Efficient Instantiation of Our Compiler 247

Bibliography 253

LIST OF FIGURES

Page

2.1 $\mathcal{F}_{\mathsf{pwKE}}$: UC symmetric PAKE functionality (original version from [48]) 11
2.2 $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE with explicit C-to-S authentication used in KHAPE 13
2.3 $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE functionality used in OKAPE 14
2.4 $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE with explicit C-to-S authentication 15

3.1 $\mathcal{F}_{\mathsf{khAKE}}$: Functionality for Key-Hiding AKE 23
3.2 Protocol $3DH$: "Triple Diffie-Hellman" Key Exchange 30
3.3 Simulator SIM showing that 3DH realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$") 32
3.4 3DH: Environment's view of real-world interaction (Game 0) 33
3.5 3DH: Environment's view of ideal-world interaction (Game 7) 36
3.6 Protocol HMQV [100] 43
3.7 Simulator SIM showing that HMQV realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$") 45
3.8 HMQV: Environment's view of real-world interaction (Game 0) 46
3.9 HMQV: Environment's view of ideal-world interaction (Game 7) 48
3.10 Protocol SKEME: KEM-authenticated Key Exchange 54
3.11 Simulator SIM showing that SKEME realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$") 55
3.12 SKEME: Environment's view of real-world interaction (Game 0) 57
3.13 SKEME: Environment's view of ideal-world interaction (Game 7) 58
3.14 Protocol KHAPE: Compiler from Key-Hiding AKE to aPAKE 64
3.15 Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 1 69
3.16 Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 2 70
3.17 Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE 72
3.18 Proof of KHAPE security: Game 4 75
3.19 KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 8) 78
3.20 KHAPE with HMQV: Concrete aPAKE protocol KHAPE-HMQV 81

4.1 Symmetric PAKE: *EKE* (a) vs. our asymmetric PAKE's (b) 90
4.2 $\mathcal{F}_{\mathsf{otkAKE}}$: Functionality for key-hiding one-time key AKE 96
4.3 otkAKE protocol $2DH$ 99
4.4 2DH: Environment's view of real-world interaction (Game 0) 101
4.5 Simulator SIM showing that 2DH realizes $\mathcal{F}_{\mathsf{otkAKE}}$ (abbreviated "$\mathcal{F}$") 102
4.6 2DH: Environment's view of ideal-world interaction 104
4.7 otkAKE protocol One-Pass HMQV 110
4.8 One-Pass HMQV: Environment's view of real-world interaction (Game 0) 112
4.9 Simulator SIM showing that protocol One-Pass HMQV realizes $\mathcal{F}_{\mathsf{otkAKE}}$ 113

4.10 One-Pass HMQV: Environment's view of ideal-world interaction (Game 7) 117
4.11 $\mathcal{F}_{\mathsf{rotkAKE}}$: Functionality for "restricted" key-hiding one-time key AKE 120
4.12 otkAKE protocol 1/2-SKEME 121
4.13 Simulator SIM showing that 1/2-SKEME realizes $\mathcal{F}_{\mathsf{rotkAKE}}$ (abbreviated "$\mathcal{F}$") . . . 122
4.14 1/2-SKEME: Environment's view of real-world interaction (Game 0) 123
4.15 1/2-SKEME: Environment's view of ideal-world interaction (Game 7) 126
4.16 Protocol OKAPE: Compiler from key-hiding otkAKE to aPAKE 131
4.17 real-world (left) vs. simulation (right) for protocol OKAPE 134
4.18 Simulator SIM showing that protocol OKAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 1 135
4.19 Simulator SIM showing that protocol OKAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 2 136
4.20 Game 0: $\mathcal{Z}$'s interaction with real-world protocol OKAPE 138
4.21 OKAPE Game 5: changes before replacement of protocol with the functionality . . 139
4.22 OKAPE Game 6: replacing the protocol with the functionality $\mathcal{F}_{\mathsf{otkAKE}}$ 141
4.23 OKAPE Game 5: delaying password file creation 144
4.24 OKAPE Game 8: Removing the kdf 146
4.25 OKAPE Game 9: rnd sessions and removing password usage 148
4.26 OKAPE Game 10: $\mathcal{Z}$'s view of ideal-world interaction 151

5.1 Left: two-round Feistel (2F) used in McQuoid et al. [109]; Right: our circuit m2F. The change from 2F to m2F is small: If $k = H'(pw, T)$, then 2F sets $s = k \oplus r$, whereas m2F sets $s = \mathsf{BC.Enc}(k, r)$, where BC is a block cipher. . . 157
5.2 Ideal functionality $\mathcal{F}_{\mathsf{RIC}}$ for *(Randomized) Half-Ideal Cipher* on $\mathcal{D} = \mathcal{R} \times \mathcal{G}$. 163
5.3 Simulator SIM for the proof of Theorem 5.1 168
5.4 The ideal-world Game 00, and its modification Game 11 (text in gray) . . . 169
5.5 Game-changes (part 1) in the proof of Theorem 5.1 170
5.6 Game-changes (part 2) in the proof of Theorem 5.1 171
5.7 Fresh queries to m2F.Dec are replaced by the circuit 175
5.8 Expanding BC.Dec 177
5.9 Replacing usage of TRIC by direct access to TBC 179
5.10 Full description of Game 8: one step away from the real-world 180
5.11 Game 9: the real-world interaction between $\mathcal{Z}$ and m2F 181
5.12 EKE: Encrypted Key Exchange with Randomized Ideal Cipher 184
5.13 Simulator SIM for the proof of Theorem 5.2 186
5.14 Game changes for the proof of Theorem 5.2 (compare Fig. 5.13 for notation) 189
5.15 EKE-KEM: Encrypted Key Exchange with Randomized Ideal Cipher (KEM version) 198
5.16 Simulator SIM for the proof of Theorem 5.3 201
5.17 Game changes for the proof of Theorem 5.3 205
5.18 protocol KHAPE using Randomized Ideal Cipher (changes from [74] marked so) . 209
5.19 Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE 210
5.20 Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$ 212
5.21 KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 8) 214
5.22 Protocol EKE-KEM of Section 5.4.1 instantiated with Saber KEM 220

6.1 Ω-method: PAKE to aPAKE compiler using Signatures [72] 227
6.2 Protocol HJK$^+$(2): PAKE to aPAKE complier using DH KEM [84] 230

6.3 Protocol APAKEM: PAKE to aPAKE compiler using CCA-secure KEM 234
6.4 Simulator SIM showing that protocol APAKEM realizes $\mathcal{F}_{\mathsf{aPAKE}}$:Part 1 239
6.5 real-world (left) vs. simulation (right) for protocol APAKEM 240
6.6 Game 0: $\mathcal{Z}$'s interaction with real-world protocol APAKEM 241
6.7 $\mathcal{Z}$'s view after Game 5 244
6.8 Key-Generation Oblivious variant of our PAKE-to-aPAKE compiler 248
6.9 A three-round UC asymmetric PAKE using compiler APAKEM instantiated with UC PAKE protocol from [67] 250
6.10 Simulator SIM showing that protocol APAKEM realizes $\mathcal{F}_{\mathsf{aPAKE}}$:Part 2 251
6.11 Game 8: $\mathcal{Z}$'s interaction with ideal-world protocol APAKEM 252

LIST OF TABLES

Page

4.1 Comparison of UC aPAKE schemes, with our schemes marked [*]: (1) f,v denote resp. fixed-base and variable-base exponentiation (expo), two-base multi-expo is counted as 1.2v, O(1) stands for significantly larger costs including bilinear maps; (2) x(C) and x(S) denote x rounds if respectively client starts or server starts, while "1" denotes a single-flow protocol; (3) EA column lists the parties that explicitly authenticate their counterparty at protocol termination. OPAQUE-HMQV appeared in [88], but above we give optimized performances characteristics due to [89]. 89

5.1 Comparison of lattice-based PAKE protocols based on bandwidth, rounds, security assumptions, security claims, and security model 223

6.1 Comparison of UC aPAKE constructions. *Comments:* [(1)]For all PAKE-to-aPAKE results we assume two-round PAKE instantiated from LWE [67, 22, 19]; [(2)]Given current LWE-based NIZK's this scheme is not more efficient than Ω-method; [(3)]Current lattice-based OPRF's are significantly more costly than KEM's; [(4)]kh-AKE stands for *key-hiding AKE*, for which there are no current lattice-based solutions; 235

ACKNOWLEDGMENTS

First and foremost, I want to express my heartfelt gratitude to my PhD advisor, Stanislaw Jarecki. Stas performed his magic show on his cryptograhy class and drew me into this amazing area, and I'm extremely fortunate to receive his mentorship. Stas's unwavering support and patience have been super helpful throughout my academic journey. Whenever I went though obstacles in my research, Stas has always been there providing guidance, support and encouragement. Under his mentorship, I not only acquired knowledge in cryptography but also became a better researcher, and I anticipate to continue to glean insights from his wisdom in the future.

I would also love to thank my committee members, Athina Markopoulou and Michael Goodrich for their constructive advice and feedback. It's my honor to have such great researchers on my PhD committee. Special thanks go to Athina, who encouraged me and gave me precious support at the beginning of my PhD.

I am indebted to my dear labmates at UC Irvine: Jiayu Xu, Tatiana Bradley, Bruno Freitas Dos Santos, Po-Chu Hsu, Apurva Rai, and Phillip Matthew Nazarian. Their friendship have enriched my experience, both professionally and personally.

I also want to thank all my co-authors: It's my great pleasure to collaborate on research and write papers with them, and without them this thesis would be impossible.

My gratitude extends to Sanjam Garg for facilitating my summer visit to UC Berkeley, and also to the remarkable researchers I had the pleasure of meeting and working with at Berkeley, including Mingyuan Wang, Aarushi Goel, Guru Vamsi Policharla, James Bartusek, Sruthi Sekar, Arka Rai Choudhuri, and Dimitris Kolonelos. Their warmth made my stay truly unforgettable.

I am immensely grateful for the friendships I have forged in Irvine during my PhD journey. To all my friends in Irvine, I extend my sincerest appreciation for their unwavering support, and I am truly blessed to have them in my life.

Last but not least, my biggest gratitude goes to my family for their love and support throughout the years. Thanks to my parents who raised me up and did their best to help me overcome the obstacles of life, whose guidance and encouragement have been a beacon of strength. Their belief in my abilities has been a constant source of motivation, for which I am profoundly grateful. I also want to thank Nicole for giving me invaluable support during my PhD.

This work is funded by National Science Foundation (NSF) award #1817143. The articles involved were previously published by Springer who gives permission to incorporate those articles into this work.

VITA

Yanqi Gu

EDUCATION

Doctor of Philosophy in Networked Systems **2024**
University of California, Irvine *Irvine, CA*

Master of Science in Networked Systems **2024**
University of California, Irvine *Irvine, CA*

Bachelor of Science in Electrical Information Engineering **2018**
Beijing University of Posts and Telecommunications *Beijing, CN*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2024**
University of California, Irvine *Irvine, California*

Research Intern **Summer 2022**
Intel *Santa Clara, California*

Research Intern **Summer 2021**
JD.com *Mountain View, California*

Data&Applied Scientist Intern **Summer 2020**
Microsoft *Redmond, Washington*

R&D Intern **Summer 2019**
Intertrust *Sunnyvale, California*

TEACHING EXPERIENCE

TA, Introduction to Discrete Mathematics (ICS 6D) **Spring 24**
University of California, Irvine *Irvine, CA*

TA, Introduction to Discrete Mathematics (ICS 6D) **Fall 23**
University of California, Irvine *Irvine, CA*

TA, Introduction to Discrete Mathematics (ICS 6D) **Winter 23**
University of California, Irvine *Irvine, CA*

TA, Projects in AI (CS 175) **Spring 22**
University of California, Irvine *Irvine, CA*

TA, Introduction to Optimization (CS 268P) **Fall 21**
University of California, Irvine *Irvine, CA*

TA, Projects in AI (CS 175) **Winter 21**
University of California, Irvine *Irvine, CA*

TA, Introduction to Optimization (CS 169/268) **Fall 19**
University of California, Irvine *Irvine, CA*

TA, Introduction to Discrete Mathematics (ICS 6D) **Winter 19**
University of California, Irvine *Irvine, CA*

PAPERS IN SUBMISSION OR UNDER REVIEW

Generic compiler from PAKE to asymmetric PAKE using KEM **2024**
Preprint

Threshold PAKE with Security against Compromise of all Servers **2024**
Preprint

REFEREED CONFERENCE PUBLICATIONS

Randomized Half-Ideal Cipher on Groups with applications to UC (a)PAKE **2023**
Theory and Applications of Cryptographic Techniques (EUROCRYPT)

Asymmetric PAKE with low computation and communication **2022**
Theory and Applications of Cryptographic Techniques (EUROCRYPT)

KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange **2021**
International Cryptology Conference (CRYPTO)

ABSTRACT OF THE DISSERTATION

New Paradigms For Efficient Password Authentication Protocols

By

Yanqi Gu

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2024

Professor Stanislaw Jarecki, Chair

In the last few years the subject of password authenticated key exchange (PAKE) protocols, particularly in the client-server setting (called *asymmetric* PAKE, or aPAKE for short), has seen renewed interest due to the weaknesses of password protocols and the ongoing standardization effort at the Internet Engineering Task Force. In particular, due to vulnerabilities in PKI systems and TLS deployment, the standard PKI-based encrypted password authentication (or "password-over-TLS") often leads to disclosure of passwords and increased exploitation of phishing techniques. Even when the password is decrypted at the correct server, its presence in plaintext form after decryption, constitutes a security vulnerability as evidenced by repeated incidents where plaintext passwords were accidentally stored in large quantities and for long periods of time even by security-conscious companies. Both problems of relying on PKI and server seeing password in clear are properly solved by PAKE protocols in the password-only setting.

While many PAKE protocols have been proposed, an interesting question is that whether there is an efficiency limit (for computation or communication) for PAKE protocols, and how to achieve that. Attempting to push such limit, this dissertation proposes a new paradigm for building efficient (a)PAKE protocols. First we present a minimal-cost aPAKE compiler called KHAPE, which offers the best performance in terms of exponentiations with less than

the cost of an exponentiation on top of an un-authenticated Diffie-Hellman key exchange. In the followup work we propose OKAPE, which further improve the round complexity of KHAPE to only two rounds of messages by leveraging a unilaterally authenticated key exchange. Since both KHAPE and OKAPE relies on Ideal Cipher (IC) Model, and existng construction for Ideal Cipher on groups all have drawbacks, we present a weakened version called Randomized Ideal Cipher (RIC) by giving up randomness of part of the ciphertext but still can be used as a drop-in replacement for IC applications. We proved that the modified 2-Feistel realizes this notion, which further improves the computation efficiency for our aPAKE compilers. Furthermore, by replacing IC with RIC in EKE protocol we get a PAKE compiler from any CPA-secure and anonymous KEM, which also opens the door for efficient lattice-based PAKE. Finally, we develop a PAKE-to-aPAKE compiler from KEM, which by embedding the previous KEM-based PAKE we can achieve an aPAKE compiler from KEM.

Chapter 1

Introduction

1.1 Problem Statement

Password, as one of the most common and prevalent ways for authentication, has been widely used in the digital world for a long time. Although password has been widely adopted in real-world authentication scenarios, there are various attacks against it. Because of our limited memory, the passwords we choose are usually of low entropy and easy to guess compared to cryptographic keys. Also we tend to reuse the same or correlated passwords. The result is that, attacker can try to guess which password we use from a dictionary of commonly used passwords, and if one of the passwords get leaked, all the applications we use which authenticated with this password can be in danger. The user side is pretty problematic. However, the server side is also not optimistic.

Password-over-TLS. The current password authentication method in practice is the PKI-based "password-over-TLS" protocol. The client will first receive a server public key verified by a Certificate Authority (CA), and then run the TLS handshake protocol and establish a secure channel between client and server, and send its password to the server through

this secure channel. On the serve side, server will receive the password in clear and hash it to verify against the stored password file, which contains a salted hash of the password, where this salt is a random value picked by server. On the first glance of this authentication process the protection here seems obvious: an attacker has to compromise the server in order to gets access to the password file, and even then the attacker cannot directly retrieve the client's password, instead the attacker is forced to run an exhaustive offline dictionary attack to find the client's password given a dictionary of candidate passwords. However, there are two disadvantages of this approach: (1) everytime the client tries to login, server can see the password in clear (and you don't want some random guy working in the service provider company to see your passwords!) and (2) the security relies on the TLS channel and breaks immediately if attacker compromises the CA company and establish a TLS channel with an adversarial public key. Today a large portion of the CA market is occupied by small and non-competent companies which are vulnerable against attackers and thus relying on CA as a trusted party is not a good choice. Many previous password leakages have happened for this reason[1][2].

Password Authenticated Key Exchange (PAKE). Password authentication protocols have been extensively studied in the cryptgraphic literature, starting from [28]. The majority of work focuses on password-only protocols where there is no assumption of any secure channel built from Public Key Infrastructure (PKI). The basic setting is modeled as Password-Authenticated Key Exchange (PAKE), where two parties only input a low-entropy password on both sides, and they will establish the same high-entropy session key if and only if their input passwords are same, otherwise they will abort or receive random different keys. The security of the PAKE protocol requires security against offline dictionary attack, i.e. against an active attacker possessing a dictionary of candidate passwords. The only allowed attack is the unavoidable online guessing attack where the adversary guesses a password and run the PAKE protocol with either party, and succeeds if adversary picks the correct password.

This above PAKE protocol is symmetric, password-only (no PKI needed!) and secure against offline dictionary attack. It has been used in many real world applications such as E-Passport. However, this symmetric setting is not suitable for the client-server setting in the Internet authentication protocols, since a compromise of the server would immediately leak all the client passwords. In an asymmetric PAKE (aPAKE) protocol, the server stores a password file, which consists of a one-way image of the password, as in the previously mentioned password-over-TLS approach. Hence, even after compromising the server and stealing the password file, the attacker still needs to perform an exhaustive offline dictionary attack as in password-over-TLS. If the password client chooses is of high-entropy, it will take a long time for this exhaustive search to succeed.

An even stronger notion of asymmetric PAKE called saPAKE further strenthens aPAKE by requiring the server to store a salted hash of the password using a private random salt. SaPAKE prevents a pre-computation attack where an attacker can pre-compute hashes of passwords from a candidate password dictionary D, and once attacker compromises server, the brute-force search for the matching password can be done in only $log|D|$ time, instead of $|D|$ which we would expect. This pre-computation attack is solved by [88] leveraging Oblivious Pseudorandom Function (OPRF) to achieve a saPAKE. In fact, one can combine any aPAKE with an OPRF to derive a saPAKE.

Efficiency Measurement. Here we use the most commonly considered efficiency measurement for cryptographic protocols, i.e. computation cost and communication cost. Our measurement on computation cost focuses on the most time consuming operations, e.g. exponentiation, hash-onto-curve, etc. In terms of communication cost, we basically count the message flows, and also the bandwidth.

1.1.1 Roadmap

In Chapter 2 we provide all the preliminary information including notations, cryptographic assumptions and cryptographic primitives. In Chapter 3 we present a minimal-cost aPAKE compiler called KHAPE, which essentially matches the computational cost of unauthenticated key exchange. In Chapter 4 we propose OKAPE, which takes only two rounds of communications while still enjoying the benefit of minimal computational cost. Both KHAPE and OKAPE rely on Ideal Cipher Model, and in Chapter 5 we further propose an efficient construction for Ideal Cipher on groups, apply it onto (a)PAKEs to achieve best performance, and open the door for efficient post-quantum PAKE. Finally, in Chapter 6 we also propose a general PAKE-to-aPAKE compiler from KEM which has efficient post-quantum instantiations.

Chapter 2

Preliminaries

2.1 Cryptographic Assumptions

For the following definitions, let g generate a cyclic group $\mathbb{G}$ of prime order p. We assume that $|p|$ is polynomial in terms of the security parameter κ.

CDH and Gap CDH. The Computational Diffie-Hellman (CDH) assumption on $\mathbb{G}$ states that given $(X, Y) = (g^x, g^y)$ for $(x, y) \overset{r}{\leftarrow} (\mathbb{Z}_p)^2$ it is hard to find $\mathsf{cdh}_g(X, Y) = g^{xy}$. The Gap CDH assumption states that CDH is hard even if the adversary has access to a Decisional Diffie-Hellman oracle ddh_g, which on input (A, B, C) returns 1 if $C = \mathsf{cdh}_g(A, B)$ and 0 otherwise.

2.2 Cryptographic Primitives

Single-round Key Exchange (KE) Scheme. A (single-round) KE scheme is a pair of algorithms $\mathsf{KA} = (\mathsf{msg}, \mathsf{key})$, where:

- msg, on input a security parameter κ, generates message M and state x;
- key, on input state x and incoming message M', generates session key K.

The correctness requirement is that if two parties exchange honestly generated messages then they both output the same session key. The KE security requirement is that a KE transcript hides the session key. Note that an additional property of KE called a *random-message* property, namely that messages output by msg are indistinguishable from values sampled from a uniform distribution over some domain $\mathcal{M}$, is required by our protocols in this work.

Definition 2.1. *KE scheme* $(\mathsf{msg}, \mathsf{key})$ *is* secure *if distributions* $\{(M_1, M_2, K)\}$ *and* $\{(M_1, M_2, K^*)\}$ *are computationally indistinguishable, where* $(x_1, M_1) \leftarrow \mathsf{msg}(1^\kappa)$*,* $(x_2, M_2) \leftarrow \mathsf{msg}(1^\kappa)$*,* $K \leftarrow \mathsf{key}(x_1, M_2)$*, and* $K^* \xleftarrow{r} \{0,1\}^\kappa$*.*

Definition 2.2. *KE scheme* $(\mathsf{msg}, \mathsf{key})$ *has the* random-message *property on domain* $\mathcal{M}$*, indexed by* κ*, if the distribution* $\{M \mid (x, M) \leftarrow \mathsf{msg}(1^\kappa)\}$ *is computationally indistinguishable from uniform over set* $\mathcal{M}[\kappa]$*.*

Key Encapsulation Mechanism. A *key encapsulation mechanism* (KEM) is a tuple of efficient algorithms $(\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$, the first two randomized, the third usually deterministic, where

- kg, on input security parameter κ, generates public key pair (sk, pk);
- enc, on input a public key pk, generates ciphertext e and session key k;
- dec, on input a private key sk and a ciphertext e, outputs a session key k.

Note that in some of our protocols, e.g. Figure 6.3, we consider KEM's where the key generation algorithm kg picks $sk \leftarrow \{0,1\}^\kappa$ and sets $pk \leftarrow \mathcal{PK}(sk)$ using a deterministic

algorithm $\mathcal{PK}$. This separation between sk choice and pk computation fits many KEM algorithms, including Kyber [40]. [1]

KEM correctness requirement is that $\Pr[\mathsf{dec}(sk, e) = k \mid (sk, pk) \leftarrow \mathsf{kg}(1^\kappa), (e, k) \leftarrow \mathsf{enc}(pk)]$ $\geq 1 - \epsilon$ where ϵ is

Definition 2.3. *KEM is* IND-CPA secure *if for every efficient algorithm $\mathcal{A}$, quantity $|p_0 - p_1|$ is a negligible function of κ, where for $i = 0, 1$ we set $p_i = \Pr[1 \leftarrow \mathcal{A}(pk, e, k_i) \mid (sk, pk) \leftarrow \mathsf{kg}(1^\kappa), (e, k_0) \leftarrow \mathsf{enc}(pk), k_1 \leftarrow \{0,1\}^\kappa]$.*

Definition 2.4. *KEM is* IND-CCA secure *if for every efficient algorithm $\mathcal{A}$, quantity $|p_0 - p_1|$ is a negligible function of κ, where for $i = 0, 1$ we set $p_i = \Pr[1 \leftarrow \mathcal{A}^{\mathsf{dec}_{sk,e}(\cdot)}(pk, e, k_i) \mid (sk, pk) \leftarrow \mathsf{kg}(1^\kappa), (e, k_0) \leftarrow \mathsf{enc}(pk), k_1 \leftarrow \{0,1\}^\kappa]$, where oracle $\mathsf{dec}_{sk,e}(\bar{e})$ returns $\mathsf{dec}_{sk}(\bar{e})$ if $\bar{e} \neq e$ and $\perp$ if $\bar{e} = e$.*

Definition 2.5. *KEM scheme has* uniform public keys *for domain $\mathcal{PK}$, indexed by the security parameter κ, if the distribution $\{pk \mid (sk, pk) \leftarrow \mathsf{kg}(1^\kappa)\}$ is computationally indistinguishable from uniform over set $\mathcal{PK}[\kappa]$.*

Definition 2.6. *KEM scheme is* weak (key-)anonymous *if distributions $\{(pk_0, pk_1, e_0)\}$ and $\{(pk_0, pk_1, e_1)\}$ are computationally indistinguishable, where $(sk_b, pk_b) \leftarrow \mathsf{kg}(1^\kappa)$ and $(e_b, k_b) \leftarrow \mathsf{enc}(pk_b)$ for $b = 0, 1$.*

Definition 2.7. *KEM scheme is* strong (key-)anonymous *if distributions $\{(sk_0, pk_0, sk_1, pk_1, e_0)\}$ and $\{(sk_0, pk_0, sk_1, pk_1, e_1)\}$ are computationally indistinguishable, where (sk_b, pk_b, e_b) for $b = 0, 1$ are chosen as in Definition 2.6.*

Definition 2.8. *KEM scheme is* OW-PCA secure *if for every efficient algorithm $\mathcal{A}$, probability $\Pr[\mathsf{dec}_{sk}(e) = k' \mid (sk, pk) \leftarrow \mathsf{kg}(1^\kappa), (e, k) \leftarrow \mathsf{enc}(pk), k' \leftarrow \mathcal{A}^{\mathsf{PCO}_{sk}(\cdot,\cdot)}(pk, e)]$ is a negligible function of κ, where oracle $\mathsf{PCO}_{sk}(\bar{e}, \bar{k})$ returns 1 if $\mathsf{dec}_{sk}(\bar{e}) = \bar{k}$, and 0 otherwise.*

[1] Every kg algorithm can be broken down into such steps if sk denotes kg randomness, but such representation of sk might not be most efficient for algorithm dec, hence in Figure 6.8 we show a version of our aPAKE protocol with black-box use of the key generation algorithm kg.

Authenticated Encryption. A (symmetric) *authenticated encryption* scheme (AE) is a tuple of efficient algorithms $(\mathsf{AEnc}, \mathsf{ADec})$, where

- AEnc, on input key $k \in \{0,1\}^\kappa$ and message $m \in \{0,1\}^*$, outputs ciphertext c;
- ADec, on input key $k \in \{0,1\}^\kappa$ and a ciphertext c, outputs $m \in \{0,1\}^* \cup \{\perp\}$.

Correctness requires that $\mathsf{ADec}(k, \mathsf{AEnc}(k,m)) = m$ for any $k \in \{0,1\}^\kappa$, $m \in \{0,1\}^*$.

Definition 2.9. *AE scheme is* IND-CCA secure *if for every efficient algorithm $\mathcal{A}$, quantity $|p_0 - p_1|$ is a negligible function of κ, where for $i = 0,1$ we set $p_i = \Pr[1 \leftarrow \mathcal{A}^{\mathsf{ADec}(k,\cdot)}(c) \mid (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{ADec}(k,\cdot)}, c \leftarrow \mathsf{AEnc}(k, m_i), k \leftarrow \{0,1\}^\kappa]$, where oracle $\mathsf{ADec}(k, \bar{c})$ returns $\mathsf{ADec}(k, \bar{c})$ if $\bar{c} \neq c$ and $\perp$ if $\bar{c} = c$.*

Definition 2.10. *AE scheme is* random-key robust *if for any efficient algorithm $\mathcal{A}$, probability $\Pr[\mathsf{ADec}(k_1, c) \neq \perp \ \wedge \ \mathsf{ADec}(k_2, c) \neq \perp \mid k_1 \leftarrow \{0,1\}^\kappa, k_2 \leftarrow \{0,1\}^\kappa, c \leftarrow \mathcal{A}(k_1, k_2)]$ is a negligible function of κ.*

Definition 2.11. *AE scheme is* unforgeable *if for any efficient algorithm $\mathcal{A}$, probability $\Pr[\mathsf{ADec}(k, c^*) \neq \perp \ \wedge \ c^* \notin \mathsf{cset} \mid k \leftarrow \{0,1\}^\kappa, c^* \leftarrow \mathcal{A}^{\mathsf{AEnc}_k(\cdot)}(1^\kappa)]$ is a negligible function of κ, where cset is the set of responses sent by oracle $\mathsf{AEnc}_k(\cdot)$.*

IND-CCA security and unforgeability properties of AE are achieved by standard AE constructions. The random-key robustness can be achieved using encrypt-then-MAC with a MAC which is collision resistant with respect to the message and the key, which can be instantiated with HMAC with full hash output. Alternatively, random-key robustness can be achieved by adding hash $\mathsf{H}(k, c)$ to an AE's ciphertext c if H is an RO hash.

Message Authentication Code. *Message authentication code* (MAC) is a tuple of efficient algorithms $(\mathsf{Mac}, \mathsf{Vrfy})$, where

- Mac, on input key $k \in \{0,1\}^\kappa$ and message $m \in \{0,1\}^*$, outputs tag t;

- Vrfy, on input key $k \in \{0,1\}^\kappa$, message m, and tag t, outputs bit $b \in \{0,1\}$.

Correctness is that $\mathsf{Vrfy}(k, m, \mathsf{Mac}(k,m)) = 1$ for any $k \in \{0,1\}^\kappa$, $m \in \{0,1\}^*$.

Definition 2.12. *MAC scheme is* unforgeable *if for any efficient algorithm $\mathcal{A}$, probability* $\Pr[\ \mathsf{Vrfy}(k, m^*, t^*) = 1\ \wedge\ m^* \notin \mathsf{mset}\ |\ k \leftarrow \{0,1\}^\kappa, (m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{Mac}(k,\cdot)}(1^\kappa)\ ]$ *is a negligible function of κ, where* mset *is the set of queries $\mathcal{A}$ sent to oracle* $\mathsf{Mac}_k(\cdot)$*.*

Definition 2.13. *MAC scheme is* tag-random *if $\forall m \in \{0,1\}^*$, distribution* $\{\mathsf{t}\ |\ k \leftarrow \{0,1\}^\kappa, \mathsf{t} \leftarrow \mathsf{Mac}(k,m)\}$ *is computationally indistinguishable from uniform over set $\mathcal{T}[\kappa]$.*

Note that the last property is satisfied if Mac is a PRF, and that many standard MAC constructions, e.g. CBC-MAC and HMAC, are PRF's.

2.3 Security Models and Frameworks

2.3.1 The Random Oracle Model and Ideal Cipher Model

We use Random Oracle Model (ROM) for proving the security claims in this study. In some cases we also assume Ideal Cipher Model (IC).

2.3.2 Universally Composability Framework

In this dissertation we use the *Universal Composability* (UC) framework [46] to construct security proofs. UC follows the simulation-based paradigm where the security of a protocol is modeled by a machine called the ideal functionality $\mathcal{F}$, which interacts with a set of

"dummy" parties and an ideal world adversary SIM, and does all computation in the ideal world. We say that protocol π securely realizes $\mathcal{F}$ if for any PPT $\mathcal{A}$, there is a simulator SIM s.t. for all environments $\mathcal{Z}$, the difference between the real-world view, i.e. an interaction of $\mathcal{Z}$ and $\mathcal{A}$ with parties executing π, and the ideal-world view, i.e. an interaction of $\mathcal{Z}$ and $\mathcal{A}$ with SIM and $\mathcal{F}$, is negligible in κ.

Note that UC framework is widely used for proving the security of PAKE protocols, and the reason is that, compared to game based security definition, UC framework supports choosing arbitrary password from any distribution, and allows reuse of these passwords. This reflects the real world application scenario.

2.3.3 Functionalities

Here we show the fundamental UC functionalities used in this work.

PAKE. Figure 2.1 shows the original version of UC symmetric PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ from [48]. Note that [72] revised symmetric PAKE functionality of [48], by extending PAKE outputs with a transcript, whereas in the original PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ of [48] the only output of each party was a session key.

In Chapter 6 we provide a variant which further simplifies [72] by requiring that the ideal-world adversary creates the session key and transcript outputs with a single command, NewKey, instead of two separate queries. This is only a syntactic difference, because real-world PAKE parties terminate with both outputs, a transcript and a session key, at the same time. Moreover, in the version in [72] the transcript-generation command is non-informational, i.e. the adversary doesn't learn anything from it, hence each PAKE protocol which is simulatable in the original GMR model must also be simulatable in ours.

aPAKE. Figure 2.4 shows the functionality $\mathcal{F}_{\mathsf{aPAKE}}$ for asymmetric PAKE with explicit

Notation: κ is the security parameter, P, P' are arbitrary parties, $\mathcal{A}$ is the ideal-world adversary

On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ from party P:

If this is the first $\mathsf{NewSession}$ query for this sid, or it is the second one and the previous one was $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw')$, then record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked fresh and forward $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}')$ to $\mathcal{A}$.

On query $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ from adversary $\mathcal{A}$:

If there is record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked fresh then:

- If $pw^* = pw$ then mark this record $\mathsf{compromised}$ and reply "correct" to S
- If $pw^* \neq pw$ then mark this record $\mathsf{interrupted}$ and reply "incorrect" to S

On query $(\mathsf{NewKey}, \mathsf{sid}, P, K^*)$ from adversary $\mathcal{A}$:

If there is record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked $\mathsf{flag} \neq \mathsf{completed}$ then:

- If $\mathsf{flag} = \mathsf{compromised}$ then set $K \leftarrow K^*$;
- If $\mathsf{flag} = \mathsf{fresh}$, there is a record $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw)$, and $\mathcal{F}_{\mathsf{pwKE}}$ sent (sid, K') to P' when record $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw)$ was fresh, then set $K \leftarrow K'$;
- In any other case set $K \xleftarrow{r} \{0,1\}^\kappa$.

Mark record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ as $\mathsf{completed}$ and send (sid, K) to P.

Figure 2.1: $\mathcal{F}_{\mathsf{pwKE}}$: UC symmetric PAKE functionality (original version from [48])

C-to-S authentication, which is used in KHAPE.

In OKAPE we include a UC aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ which is different from KHAPE[74], shown in Figure 2.3. This functionality is largely as it was originally defined by Gentry, Mackenzie, and Ramzan [72], and it adopts few notational modifications introduced by Figure 2.4. These include naming what amounts to user accounts explicitly as uid instead of generic-sounding sid, using sid instead of ssid as a session-identifier for on-line authentication attempts, and using only pairs $(\mathsf{S}, \mathsf{uid})$ to identify server password files and not $(\mathsf{S}, \mathsf{U}, \mathsf{uid})$ tuples as in [72].

Because we differentiate between unsalted and (publicly) salted aPAKE's, an explicit support for unsalted aPAKE's is reflected in aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ by introducing a slight modification in the functionality of [74]. These modifications are highlighted in Figure 2.3, and they all concern a client-side usage of the user account field uid. When the aPAKE protocol is *unsalted*, to enforce the aPAKE contract defined by [72], which is that a single real-world offline dictionary attack operation must correspond not only to a single password guess but also to a unique user password file, identified by a unique pair $(\mathsf{S}, \mathsf{uid})$, the client must get as environment's inputs both the server identifier S *and* the user account identifier uid. This is reflected in including uid in the inputs to $\mathsf{CltSession}$ command in Figure 2.3. However, since the client now performs computation on a fixed uid, honest client and server sessions will not agree on the same output key unless they run not only on the same password pw but also on the same uid. Hence the NewKey processing now includes uid-equality enforcement. Finally, for the same reason, an online password test $\mathsf{TestPwd}$ must specify the uid field in addition to password guess pw^*.

Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ in Figure 2.3 currently allows both the server and the client sessions to leak the account identifier uid input to the adversary. The server-side leakage of this information was inherent (although not immediate to observe) in the original aPAKE functionality of [72], and it was adopted by subsequent works, including e.g. [88, 74]. Now, however, we

Password Registration

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$ from S create record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked fresh.

Stealing Password Data

- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is no record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, return "no password file". Otherwise mark this record $\mathsf{compromised}$, and if there is a record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ then send pw to $\mathcal{A}$.

- On $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ from $\mathcal{A}$, then do:

  - If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$, do the following:
    If $pw^* = pw$ then return "correct guess" to $\mathcal{A}$ else return "wrong guess."
  - Else record $\langle \mathsf{offline}, \mathsf{S}, uid, pw^* \rangle$

Password Authentication

- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ from C, if there is no record $\langle \mathsf{sid}, \mathsf{C}, ... \rangle$ then record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh and send $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$.

- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ from S, if there is no record $\langle \mathsf{sid}, \mathsf{S}, ... \rangle$ then retrieve record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, and if it exists then create record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, pw, 1 \rangle$ marked fresh and send $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ to $\mathcal{A}$.

Active Session Attacks

- On $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ marked fresh, then do: If $pw^* = pw$ then mark it $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

- On $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh, then do: If there is a record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$ then mark $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if there is a record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ not marked $\mathsf{completed}$, then do:

  - If rec is marked $\mathsf{compromised}$ set $K \leftarrow K^*$;
  - Else if $\mathsf{role} = 0$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 1 \rangle$ s.t. $\mathcal{F}_{\mathsf{aPAKE}}$ sent (sid, K') to P' while that record was marked fresh, set $K \leftarrow K'$;
  - Else if $\mathsf{role} = 1$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 0 \rangle$ which is marked fresh, pick $K \xleftarrow{r} \{0,1\}^\ell$;
  - Else set $K \leftarrow \perp$.

  Finally, mark rec as $\mathsf{completed}$ and send output (sid, K) to P.

Figure 2.2: $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE with explicit C-to-S authentication used in KHAPE

Password Registration

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$ from S create record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked fresh.

Stealing Password Data [*these queries must be approved by the environment*]

- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is no record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, return "no password file". Otherwise mark this record $\mathsf{compromised}$, and if there is a record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ then send pw to $\mathcal{A}$.

- On $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ from $\mathcal{A}$, then do:

  - If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$, do the following:
    If $pw^* = pw$ then return "correct guess" to $\mathcal{A}$ else return "wrong guess."
  - Else record $\langle \mathsf{offline}, \mathsf{S}, uid, pw^* \rangle$

Password Authentication

- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, \boxed{\mathsf{uid}}, pw)$ from C, if there is no record $\langle \mathsf{sid}, \mathsf{C}, ... \rangle$ then save $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \boxed{\mathsf{uid}}, pw, 1 \rangle$ marked fresh, send $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \boxed{\mathsf{uid}})$ to $\mathcal{A}$.

- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ from S, if there is no record $\langle \mathsf{sid}, \mathsf{S}, ... \rangle$ then retrieve record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, and if it exists then save $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \boxed{\mathsf{uid}}, pw, 2 \rangle$ marked fresh and send $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ to $\mathcal{A}$.

Active Session Attacks

- On $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \boxed{\mathsf{uid}}, pw^*)$ from $\mathcal{A}$, if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', \boxed{\mathsf{uid}}, pw, \mathsf{role} \rangle$ marked fresh, then do: If $pw^* = pw$ then mark it $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

- On $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \boxed{\mathsf{uid}}, pw, 1 \rangle$ marked fresh, then do: If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$ then mark rec $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', \boxed{\mathsf{uid}}, pw, \mathsf{role} \rangle$ not marked $\mathsf{completed}$, then do:

  1. If rec is marked $\mathsf{compromised}$ set $K \leftarrow K^*$;
  2. Else if rec is fresh and there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, \boxed{\mathsf{uid}}, pw, \mathsf{role}' \rangle$ for $\mathsf{role}' \neq \mathsf{role}$ and $\mathcal{F}_{\mathsf{aPAKE}}$ sent (sid, K') to P' when this record was fresh, set $K \leftarrow K'$;
  3. Else set $K \xleftarrow{r} \{0,1\}^{\ell}$.

  Finally, mark rec as $\mathsf{completed}$ and send output (sid, K) to P.

  Note: Modifications from $\mathcal{F}_{\mathsf{aPAKE}}$ *defined in [74] are marked* $\boxed{\textit{like this}}$. *They consist of assumping input* uid *in* CltSession *and* TestPwd *and enforcing* uid*-equality between client and server sessions in* NewKey *processing.*

Figure 2.3: $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE functionality used in OKAPE

Queries $\mathsf{StorePwdFile}$ from S, $\mathsf{StealPwdFile}$ or $\mathsf{OfflineTestPwd}$ from $\mathcal{A}$, $\mathsf{CltSession}$ from C, $\mathsf{SvrSession}$ from S, and $\mathsf{TestPwd}$ or $\mathsf{Impersonate}$ from $\mathcal{A}$, functionality $\mathcal{F}_{\mathsf{aPAKE}}$ acts as $\mathcal{F}_{\mathsf{aPAKE}}$ of Figure 2.3, *except* it omits all parts marked uid (i.e. it does not require uid input for C and does not enforce uid-equality for C and S).

Below we mark like this parts of NewKey processing which differ from $\mathcal{F}_{\mathsf{aPAKE}}$.

Key Generation and Authentication

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if there is a record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role}\rangle$ not marked $\mathsf{completed}$, then do:

  1. If rec is marked $\mathsf{compromised}$ set $K \leftarrow K^*$;
  2. Else if rec is fresh, $\mathsf{role} = 2$, and there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 1 \rangle$ s.t. $\mathcal{F}_{\mathsf{aPAKE}}$ sent (sid, K') to P' when this record was fresh, set $K \leftarrow K'$;
  3. Else if $\mathsf{role} = 1$ set $K \xleftarrow{r} \{0,1\}^\ell$, and if $\mathsf{role} = 2$ set $K \leftarrow \perp$.

  Finally, mark rec as $\mathsf{completed}$ and send output (sid, K) to P.

Figure 2.4: $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE with explicit C-to-S authentication

also introduce client-side leakage of the same information. The uid has to be transmitted from the client to the server before the protocol starts, but it is not clear that the cryptographic protocol should leak it. We leave plugging this leakage and/or verifying whether it is necessary in known aPAKEs, including ours, to future work.

Client-to-server entity authentication. Since protocol OKAPE includes client-to-server authentication (it is *not* optional, and the protocol is insecure without it), it realizes an aPAKE functionality amended by client-to-server entity authentication. We use $\mathcal{F}_{\mathsf{aPAKE}}$ to denote the variant of aPAKE functionality with uni-directional client-to-server entity authentication, and we include it in Figure 2.4. Since protocol OKAPE is a *salted* aPAKE, it does not need the uid input on the client side, so the $\mathcal{F}_{\mathsf{aPAKE}}$ functionality in Figure 2.4 incorporates all the code of functionality $\mathcal{F}_{\mathsf{aPAKE}}$ but without the uid-related modifications. To simplify NewKey processing functionality $\mathcal{F}_{\mathsf{aPAKE}}$ in Figure 2.4 assumes that the client party terminates first, so if two honest parties are connected then the client party computes its session key output first, and it is always the server party which can potentially get the same key copied by the functionality. One could define it more generally but we expect that

in most aPAKE protocols with unilateral client-to-server explicit authentication the server will indeed be the last party to terminate.

Chapter 3

KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange

3.1 Introduction

In this paper we investigate the question of *how "minimal" an asymmetric PAKE can be.* In spite of the many subtleties surrounding the design and analysis of aPAKE protocols, there are several efficient and practical realizations which meet a universally composable (UC) notion of aPAKE [72]. For example, the overhead of the recently analyzed SPAKE2+ protocol [119] over the *unauthenticated* Diffie-Hellman (uDH) protocol is 1 or 2 exponentiations per party. Similar overhead costs are also imposed by the generic results which compile any PAKE to aPAKE [72, 84]. Known *strong* aPAKEs (see below), add similar or larger overhead costs [88, 44].

The comparison to uDH is significant not only from a practical point of view, but also because PAKE protocols imply unauthenticated key exchange in the sense of the Impagliazzo-Rudich results [85, 77]. Thus, we can see uDH as the lowest possible expected performance of PAKE

protocols. But how close to the uDH cost can we get; can one improve on existing protocols?

In the symmetric PAKE case, where the two peers share the same password, there are almost optimal answers to this question. The Bellovin-Merrit's classical EKE protocol [28], shows that all you need is to apply a symmetric-key encryption on top of the uDH transcript. It requires a carefully chosen encryption scheme, e.g., one that is modeled after an ideal cipher, but it only involves symmetric key techniques [26, 7, 43, 109].[1]

Can this low overhead relative to uDH be achieved also in the more involved setting of asymmetric PAKEs, where security against offline attacks is to be provided even when the server is broken into? We show an aPAKE protocol, KHAPE, that only requires symmetric operations (in the ideal cipher model) over regular authenticated DH.

KHAPE (for Key-Hiding Asymmetric PakE) can be seen as a variant of the OPAQUE protocol [88] that is being developed into an Internet standard [101] and intended for use within TLS 1.3 [120]. OPAQUE introduces the idea of password-encrypted credentials containing an encrypted private key for the user and an authenticated public key for the server. The user deposits the encrypted credentials at the server during password registration and it retrieves them for login sessions, thus allowing user and server to run a regular authenticated key exchange (AKE) protocol. However, encrypting and authenticating credentials with a password opens the protocol to trivial offline dictionary attacks. Therefore, OPAQUE first runs an Oblivious PRF (OPRF) on the user's password in order to derive a strong encryption key for the credential. This makes the protocol fully reliant on the strength of the OPRF. If OPRF is ever broken (by cryptanalysis, quantum attacks or security compromise), the user's password is exposed to an offline dictionary attack.

Near-optimal aPAKE. KHAPE addresses this weakness by dispensing with the OPRF

[1]Several other symmetric PAKE protocols, e.g. SPAKE2 [10], SPEKE [86, 105, 79] and TBPEKE [113], attain universally composable security without relying on an ideal cipher but incur additional exponentiations over uDH costs [4].

(hence also improving performance). It uses a "paradoxical" mechanism that allows to directly encrypt credentials with the password and still prevent dictionary attacks. Two key ideas are: (i) dispense with authentication of the credentials[2] and instead use a non-committing encryption where decryption of a given ciphertext under different keys cannot help identify which key from a candidate set was used to produce that ciphertext; and (ii) using a *key-hiding* AKE. The latter refers to AKE protocols that require that no adversary, not even active one, can identify the long-term keys used by the peers to an exchange even if provided with a list of candidate keys (a notion reminiscent of key anonymity for public key encryption [24]).

Fortunately, many established AKE protocols are key hiding, including implicitly authenticated protocols such as 3DH [107] and HMQV [100], and KEM-based protocols with key-hiding KEMs (e.g., SKEME [97]). The non-committing property of encryption models symmetric encryption as an ideal model (similarly to the case of EKE discussed above) and allows for implementations based on random oracles with hash-to-curve operations to encode group elements as strings. As a result, KHAPE with HMQV, uses only one fixed-base exponentiation, one variable-base (multi)exponentiation for each party, and one hash-to-curve operation for the client. In all, it achieves computational overhead relative to *unauthenticated* Diffie-Hellman of *less than the cost of one exponentiation,* thus providing a close-to-optimal answer to our motivating questions above. Such computational performance compares favorably to that of other efficient aPAKE protocols such as SPAKE2+ and OPAQUE that incur overhead of one and two (variable-base) exponentiations, respectively, for server and client. In terms of number of messages, KHAPE uses 4 (3 if server initiates), compared to 3 messages in SPAKE2+ and OPAQUE.

Refer to Section 6.2 for a detailed description and rationale of the generic KHAPE protocol (compiling any key-hiding AKE into an aPAKE) and to Section 3.7 for the instantiation

[2] Dispensing with authentication of credentials in OPAQUE completely breaks the protocol, allowing for trivial offline dictionary attacks.

using HMQV.

On Strong aPAKE and reliance on OPRF. In the comparisons above, it is important to stress that OPAQUE achieves a *stronger notion of aPAKE*, the so called Strong aPAKE (saPAKE) model from [88]. In this model, the attacker that compromises a server can only start running an offline dictionary attack after breaking into the server. In contrast, in regular aPAKE, an offline attack is still needed but a specialized dictionary can be prepared ahead of time and used to find the password almost instantaneously when breaking into the server. KHAPE, as discussed above, does not provide this stronger security. However, as shown in [88], one can add a run of an OPRF to any aPAKE protocol to achieve Strong aPAKE security. If one does that to KHAPE, one gets a Strong aPAKE protocol with performance similar to that of OPAQUE (using HMQV or 3DH).

However, there is a significant difference in the reliance on the security of OPRF. While the password security of OPAQUE breaks down with a compromise of the OPRF key (namely, it allows for an offline dictionary attack on the password), in KHAPE the effect of compromising the OPRF is only to fall back to the (non-strong) aPAKE setting. In particular, this distinction is relevant in the context of quantum-safe cryptography as there are currently no known efficient OPRFs considered to be quantum safe. This opens a path to quantum-safe aPAKEs based on KHAPE with key hiding quantum-safe KEMs.

Closer comparison with OPAQUE. As stated above, KHAPE has an advantage over OPAQUE in terms of security due to its weaker reliance on OPRF and its computational advantage when the OPRF is not used. Also, KHAPE seems more conducive to post-quantum security via post-quantum key-hiding KEMs.[3] On the other hand, KHAPE requires one more message and allows for a more restrictive family of AKEs relative to OPAQUE (e.g., it does not allow for signature-based protocols as those based on SIGMA [98] and used in TLS 1.3

[3]We are currently investigating the use of NIST's post-quantum KEM selections [112] in conjunction with KHAPE.

and IKEv2). KHAPE also relies for its analysis on the ideal cipher model while OPAQUE uses the random oracle model. An interesting advantage of KHAPE over OPAQUE is that in OPAQUE, an online attacker testing a password learns whether the password was wrong before the server does (in KHAPE the server learns first). This leads to a more complex mechanism for counting password failures at a server running OPAQUE, especially in settings with unreliable communication. Finally, we point out an advantage of using an OPRF with KHAPE (in addition to providing Strong aPAKE security): It allows for multi-server security via a threshold OPRF [87] where an attacker needs to break into multiple servers before it can run an offline attack on a password.

UC model analysis of (key-hiding) AKE's. All our protocols are framed and analyzed in the Universally Composable (UC) model [47]. This includes a formalization of the key-hiding AKE functionality that underlies the design of KHAPE. In order to instantiate KHAPE with specific AKE protocols, we prove that protocols 3DH [107] and HMQV [100] realize the key-hiding AKE functionality (in the ROM and under the Gap CDH assumption). We prove a similar result for SKEME [97] with appropriate KEM functions. We see the security analysis of these AKE protocols in the UC model, with and without key confirmation, as a contribution of independent interest. Moreover, the study of key-hiding AKE has applicability in other settings, e.g., where a gateway or IP address hides behind it other identities; say, a corporate site hosting employee identities or a web server aggregating different websites.

3.2 The Key-Hiding AKE UC Functionality

Protocol KHAPE results from the composition of an encrypted credentials scheme and a *key-hiding* AKE protocol. Fig. 3.1 defines the UC functionality $\mathcal{F}_{\mathsf{khAKE}}$ that captures the properties required from a key-hiding AKE protocol. The modeling choices target the fol-

lowing requirements: First, as shown in Section 6.2, the security and key-hiding properties of this key-hiding AKE model suffice for our main application, a generic construction of UC aPAKE from any protocol realizing $\mathcal{F}_{\mathsf{khAKE}}$. Second, adding a standard key confirmation to any protocol that realizes $\mathcal{F}_{\mathsf{khAKE}}$ results in a (standard) UC AKE with explicit entity authentication. Lastly, this functionality is realized by several well-known and efficient AKE protocols, including 3DH and HMQV, as shown in Sections 3.3 and 4.2.2, as well as by a KEM-based AKE such as SKEME, if instantiated with a key-hiding KEM, see Section 3.5. We provide more details and rationale for the $\mathcal{F}_{\mathsf{khAKE}}$ next.

High-level requirements for key-hiding AKE. The most salient property we require from AKE is *key hiding*. To illustrate this requirement consider an experiment where the attacker $\mathcal{A}$ is provided with a transcript of a session between a party P and its counterparty CP. Party P has two inputs in this AKE instance: a public key pk_{CP} for CP and its own private key K_{P} which P uses to authenticate to CP who presumably knows P's public key pk_{P}. In addition, $\mathcal{A}$ is given a pair of private keys: P's private key K_{P} and a second random independent private key. $\mathcal{A}$'s goal is to decide which of the two keys P used in that session.[4] We are interested in AKE protocols where the attacker has no better chance to answer correctly than guessing randomly *even for sessions in which $\mathcal{A}$ is allowed to choose the messages from* CP.

The key hiding property will come up in the analysis of KHAPE as follows. The attacker learns a ciphertext c that encrypts the user's private key under the user's password. By decrypting this ciphertext under all passwords in a dictionary, the attacker obtains a set of possible private keys for the user. The key hiding property ensures that the attacker cannot identify the correct key (or the password) in the set. Fortunately, as we prove here, a large class of AKE protocols satisfy the key-hiding property, including implicitly authenticated protocols such as HMQV and 3DH, and some KEM-based protocols.

[4]This is reminiscent of key anonymity for PK encryption [24] where the attacker needs to distinguish between public keys for a given ciphertext.

- PK stores all public keys created via Init;
- PK_{P} stores the public keys created by P;
- CPK stores the compromised keys;

Keys: Initialization and Attacks

On Init from P:

Send $(\mathsf{Init}, \mathsf{P})$ to $\mathcal{A}$, let $\mathcal{A}$ specify pk s.t. $pk \notin PK$, add pk to PK and to PK_{P}, and output (Init, pk) to P. If P is corrupt then add pk to CPK.

On $(\mathsf{Compromise}, \mathsf{P}, pk)$ from $\mathcal{A}$:

If $pk \in PK_{\mathsf{P}}$ then add pk to CPK.

Login Sessions: Initialization and Attacks

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ from P:

If $pk_{\mathsf{P}} \in PK_{\mathsf{P}}$ and there is no prior session record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot \rangle$ then:

- create session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp \rangle$ marked fresh;
- initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$;
- send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ to $\mathcal{A}$.

On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$:

If there is session $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \perp \rangle$ marked fresh then change it to $\mathsf{interfered}$.

Login Sessions: Key Establishment

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ from $\mathcal{A}$:

If $\exists$ session record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp \rangle$ then:

- if rec is marked fresh: If $\exists$ record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk_{\mathsf{CP}}, pk_{\mathsf{P}}, k' \rangle$ marked fresh s.t. $k' \neq \perp$ then set $k \leftarrow k'$, else pick $k \xleftarrow{\text{r}} \{0,1\}^\kappa$;
- if rec is marked $\mathsf{interfered}$ then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$;
- update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P.

Session-Key Query

On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$ from $\mathcal{A}$:

If $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \ldots \rangle$ and $pk' \notin (PK \setminus CPK)$ then send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$.

Figure 3.1: $\mathcal{F}_{\mathsf{khAKE}}$: Functionality for Key-Hiding AKE

Additionally, $\mathcal{F}_{\mathsf{khAKE}}$ strengthens the basic guarantees of AKE protocols in several ways. It requires resilience to KCI (key-compromise impersonation) attacks, namely, upon the compromise of the private key of party P, the attacker can impersonate P to others but it cannot impersonate others to P. In the aPAKE setting, this ensures that an attacker that compromises a server, cannot impersonate the client to the server without going through an offline dictionary attack. In the context of key hiding AKE, we also need KCI resilience to prevent the attacker from authenticating to the client when given a set of possible private keys for that client.

Second, $\mathcal{F}_{\mathsf{khAKE}}$ requires that keys exchanged by a honest P with a corrupted CP still maintain a good amount of randomness, namely, the attacker can cause them to deviate from uniform but not by much (a property sometimes referred to as "contributive" key exchange, and not required in standard UC treatment). In the setting of protocol KHAPE, adversarial choice of session keys (particularly the ability of the attacker to create equal keys in different sessions) could lead to protocols where the attacker can test more than one password in a single session.

Properties that we do *not* consider as part of the $\mathcal{F}_{\mathsf{khAKE}}$ functionality, but will be provided by our final aPAKE protocol, KHAPE, include key confirmation, explicit authentication and full forward secrecy ($\mathcal{F}_{\mathsf{khAKE}}$ itself implies forward secrecy only against passive attackers).

Identities and public keys. We consider a setting where each party P has multiple public keys in the form of arbitrary handles pk. In the security model we assume that the public keys are arbitrary bitstrings chosen without loss of generality by the attacker (ideal adversary) $\mathcal{A}$, with the limitation that honest parties are assigned non-repeating pk strings. Pairs (P, pk) act as regular UC identities from the environment's point of view, but the pk component is concealed from $\mathcal{A}$ during key exchange sessions, even for sessions which are actively attacked by $\mathcal{A}$. This model can capture practical settings where P represents a gateway or IP address

behind which other identities reside, e.g., a corporate site hosting employee identities or a web server aggregating different websites, and where one is interested to hide which party behind the gateway is communicating in a given session. Our specific application setting when using key-hiding AKE in the aPAKE construction of Section 6.2, is more abstract: The party symbols P, CP represent parties like internet clients and servers, while the multiplicity of public keys comes from decryptions of encrypted credentials under multiple password.

$(\mathsf{Compromise}, \mathsf{P}, pk)$. This adversarial action hands the (long-term) private key of party (P, pk) to the attacker $\mathcal{A}$. Such private-key leakage does not provide $\mathcal{A}$ with control over party P, and it does not even imply that the sessions which party P runs using the (leaked) key pk are insecure. However, when combined with the ability to run active attacks, via the $\mathsf{Interfere}$ action below, $\mathcal{A}$ can fully impersonate (P, pk) in sessions of $\mathcal{A}$'s choice. The leakage of the private key K corresponding to (P, pk) does not affect the security of a session executed by party P even if it uses the compromised key pk. This captures the KCI property, i.e. that leakage of the private key of party P does not allow to impersonate others to party P. Also, any party P' which runs AKE with a *counterparty* identity specified as (P, pk), will also be secure as long as $\mathcal{A}$ does not actively interfere in that protocol. This captures the requirement that passively-observed AKE instance are secure regardless of the compromise of the long-term secrets used by either party. Note that $\mathcal{A}$ cannot compromise a party P but rather an identity pair (P, pk) and such compromise does not affect other pairs (P, pk').

$\mathsf{NewSession}$. A session is initiated by a party P that specifies its own identity pair (P, pk) as well as the intended counterparty identity pair $(\mathsf{CP}, pk_{\mathsf{CP}})$. Session identifiers sid are assumed to be unique within an honest party. The role of the initialized session-specific random function $R_{\mathsf{P}}^{\mathsf{sid}}$ is described below. A record for a session is initialized as fresh and is represented by a tuple $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ where the last position, set to $\perp$, is reserved for recording the session key. An *essential element* in $\mathsf{NewSession}$ is that $\mathcal{A}$ learns $(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ but *it does not learn* (pk, pk_{CP}). In the real world this translates into the inability of the

attacker to identify public (or private) keys associated to a pair of parties $(\mathsf{P}, \mathsf{CP})$ engaging in the Key-Hiding AKE protocol.

The functionality enforces that an honest P can start a session only on key pk which P generated and for which it holds a private key. However, the functionality does not check anything about the intended counterparty's identity $(\mathsf{CP}, pk_{\mathsf{CP}})$, so the private key corresponding to pk_{CP} could be held by party CP, or it could be held by a different party, or it could be compromised by the adversary, or it could be that pk_{CP} was not even generated by the key generation interface of $\mathcal{F}_{\mathsf{khAKE}}$, and it is an *adversarial* public key, whose private key the environment gave to the adversary. Our model thus includes honest parties who are tricked to use a wrong public key for the counterparty (e.g., via a phishing attack) in which case the attacker may know the corresponding private key. Note that regardless of what key pk_{CP} the session runs on, it is not given to the adversary, so if it is a key created by the envriment (i.e. a higher-level application which uses the key-hiding AKE) it does not necessarily follow that this key will be known to the adversary, and only in the case it is known the adversary will be able to attack that session using interfaces Interfere, NewKey, and ComputeKey below.

Function $R_{\mathsf{P}}^{\mathsf{sid}}$. When command NewSession creates a session for $(\mathsf{sid}, \mathsf{P})$ the functionality initializes a *random* function $R_{\mathsf{P}}^{\mathsf{sid}}$ specific to this session. Function $R_{\mathsf{P}}^{\mathsf{sid}}$ is used to set the value of the session key for sessions in which $\mathcal{A}$ actively interferes. It also allows $\mathcal{A}$ to have limited control over the value of the key under strict circumstances, namely it must know the pulic keys pk, pk_{CP} used on that session, and it must compromise party $(\mathsf{CP}, pk_{\mathsf{CP}})$. Even then the only freedom $\mathcal{A}$ has is to evaluate function $R_{\mathsf{P}}^{\mathsf{sid}}$ on any point α via a ComputeKey query, see below, and then choose one such point in the NewKey caommand. This captures the "contributive" property discussed above: If an honest party runs the AKE protocol even with adversary as a counterparty, the adversary's influence over the session key is limited to pre-computing polynomially-many random key candidates and then choosing one of them

as a key on that session. The exact mechanics and functionality of $R_{\mathsf{P}}^{\mathsf{sid}}$ are defined in the NewKey and ComputeKey actions below.

$(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$. This action represents an active attack on session $(\mathsf{P}, \mathsf{sid})$ and makes the session change its status from fresh to interfered. The adversary does not have to know either P's own key pk or the intended counterparty key pk_{CP} which P uses on that session.[5] Such active atack will prevent session $(\mathsf{P}, \mathsf{sid})$ from establishing a secure key with any other honest party session, e.g. $(\mathsf{CP}, \mathsf{sid})$. It will also allow $\mathcal{A}$ to learn and/or influence the value of the session key this session outputs (using function $R_{\mathsf{P}}^{\mathsf{sid}}$), but only if in addition to being active $\mathcal{A}$ compromises the counterparty key $(\mathsf{CP}, pk_{\mathsf{CP}})$ used on session $(\mathsf{P}, \mathsf{sid})$.

NewKey. This action finalizes an AKE instance and makes $(\mathsf{P}, \mathsf{sid})$ output a session key. If the session is fresh then it receives either a fresh random key or the same key that was previously received by a matching session. If the session is interfered, the value of the session key is determined by the function $R_{\mathsf{P}}^{\mathsf{sid}}$ on input $(pk, pk_{\mathsf{CP}}, \alpha)$ where α is chosen arbitrarily by $\mathcal{A}$, allowing $\mathcal{A}$ to influence the value of the session key (but in a very limited way as explained above). In the real-world, α represents transcript elements generated by the attacker, e.g., value Y an adversarial P_2 sends to an honest party P_1 in 3DH or HMQV.

ComputeKey. This action allows $\mathcal{A}$ to query the function $R_{\mathsf{P}}^{\mathsf{sid}}$ associated to a session $(\mathsf{sid}, \mathsf{P})$, potentially allowing $\mathcal{A}$ to learn and/or influence the session key for $(\mathsf{sid}, \mathsf{P})$. Note that learning any values of function $R_{\mathsf{P}}^{\mathsf{sid}}$ is useless unless the adversary actively attacks session $(\mathsf{sid}, \mathsf{P})$, because otherwise $R_{\mathsf{P}}^{\mathsf{sid}}$ is not used to determine the key output by session $(\mathsf{sid}, \mathsf{P})$. Moreover, $\mathcal{A}$ needs to provide $(pk, pk_{\mathsf{CP}}, \alpha)$ as input to ComputeKey, and if those inputs do not match P's own key pk and the intended counterparty key pk_{CP} which P uses on session $(\mathsf{sid}, \mathsf{P})$, then this query reveals an irrelevant value, since $R_{\mathsf{P}}^{\mathsf{sid}}$ is a random fuction.

[5]Currently functionality $\mathcal{F}_{\mathsf{khAKE}}$ assumes the ideal-world adversary $\mathcal{A}$ knows, and indeed creates, all honest parties' public keys. A tighter model is possible, if $\mathcal{F}_{\mathsf{khAKE}}$ samples public keys on behalf of honest players using the prescribed key generation algorithm, instead of letting $\mathcal{A}$ pick them. This would allow modeling use cases where the public keys are not public and are not freely available to the adversary.

Finally, $\mathcal{F}_{\mathsf{khAKE}}$ releases value $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk_{\mathsf{CP}}, \alpha)$ to $\mathcal{A}$ only if key pk_{CP} is either compromised or adversarial. Summing up, the ability to learn (and/or control via the NewKey interface) the session key output by session $(\mathsf{sid}, \mathsf{P})$ is restricted to the case where all of the following hold: $\mathcal{A}$ actively interfered on that session, $\mathcal{A}$ guesses keys pk, pk_{CP} which this session uses, and $\mathcal{A}$ compromises counterparty's key $(\mathsf{CP}, pk_{\mathsf{CP}})$.

How $\mathcal{F}_{\mathsf{khAKE}}$ ensures key hiding and session security. The description of $\mathcal{F}_{\mathsf{khAKE}}$ is now complete. We now explain how $\mathcal{F}_{\mathsf{khAKE}}$ ensures the key hiding property by which $\mathcal{A}$ cannot learn the value pk for an identity pair (P, pk) even if $\mathcal{A}$ knows P, has a list of all possible values of (P, pk), and actively interacts with (P, pk) using a compromised party $(\mathsf{CP}, pk_{\mathsf{CP}})$. Let's assume these conditions hold. Note that the only actions in which $\mathcal{A}$ can learn pk values from $\mathcal{F}_{\mathsf{khAKE}}$ are upon key generation and via the ComputeKey call. Key generation assumes that $\mathcal{A}$ has a list of all possible values (P, pk). As we explain above, the only argument on which the value of function $R_{\mathsf{P}}^{\mathsf{sid}}$ is useful is a tuple $(pk, pk_{\mathsf{CP}}, \alpha)$ which the functionality uses to derive a session key for an actively attacked session $(\mathsf{sid}, \mathsf{P})$.

Consequently, the only way $\mathcal{F}_{\mathsf{khAKE}}$ can leak the session key output by $(\mathsf{sid}, \mathsf{P})$ is if $\mathcal{A}$ satisfies the three conditions above, i.e. it interferes in that session, key pk_{CP} used on that session is either compromised or adversarial, and $\mathcal{A}$ queries ComputeKey on the proper keys pk, pk_{CP}. This is also the only way $\mathcal{A}$ can learn anything about keys pk, pk_{CP} used by session $(\mathsf{sid}, \mathsf{P})$: It has to attack the session, compromise pk_{CP}, get a session key candidate k^* via query ComputeKey on pk, pk_{CP}, and then compare this key candidate against any information it has about the key k output by session $(\mathsf{sid}, \mathsf{P})$. For example, if P's higher-level application uses key k to MAC or encrypt a message, the adversary can verify the result against a candidate key k^* and thus learn whether $k^* = k$, and hence whether keys pk, pk_{CP} which $\mathcal{A}$ used to compute k^* were the same keys that were used by session $(\mathsf{sid}, \mathsf{P})$.

3.3 3DH as Key-Hiding AKE

We show that protocol 3DH, presented in Figure 3.2, realizes the UC notion of Key-Hiding AKE, as defined by functionality $\mathcal{F}_{\mathsf{khAKE}}$ in Section 4.2, under the Gap CDH assumption in ROM. As a consequence, 3DH can be used to instantiate protocol KHAPE in a simple and efficient way.

3DH [107] is a simple, implicitly authenticated key exchange used as the basis of the X3DH protocol [108] that underlies the Signal protocol. It consists of a plain Diffie-Hellman exchange authenticated via the session-key derivation that combines the ephemeral and long-term key of both peers. Specifically, if (a, A) and (b, B) are the long-term key pairs of two parties P_1 and P_2, and (x, X) and (y, Y) are their ephemeral DH values, then 3DH combines these key pairs to compute a (hash of) the *triple* of Diffie-Hellman values, $\sigma = g^{xb} \| g^{ay} \| g^{xy}$. Security of 3DH is intuitively easy to see: It follows from the fact that to compute σ the attacker must either (1) know (x, a) to attack party P_2 who uses A as a public key for its counterparty, or (2) know (y, b) to attack party P_1 who uses B as a public key for its counterparty. In other words, the attacker wins only if it is an active man-in-the-middle attacker *and* it compromises the key used as counterparty's public key by the attacked party. (Recall that "compromising a public key" stands for learning the corresponding private key.) The key-hiding property comes from the fact that the values X and Y exchanged in the protocol do not depend on long-term keys, and the fact that the only information about the long-term keys used by any party can be gleaned only from the session key they output and from H oracle queries on a σ value computed using these keys. The formal proof of key-hiding in the UC model captures this argument, and we present it below.

We note that 3DH is not the most efficient key-hiding AKE. 3DH costs one fixed-base and three variable-base exponentiations per party, and in Section 4.2.2 we will show that HMQV, which preserves the bandwidth and round complexity of 3DH but folds the three variable-

base exponentiations of 3DH into a single multi-exponentiation, realizes the key-hiding AKE functionality under the same Gap CDH assumption (although with worse exact security guarantees). However, HMQV can be seen as a modification of 3DH, and the security analysis of 3DH we show below will form a blueprint for the analysis of HMQV in Section 4.2.2.

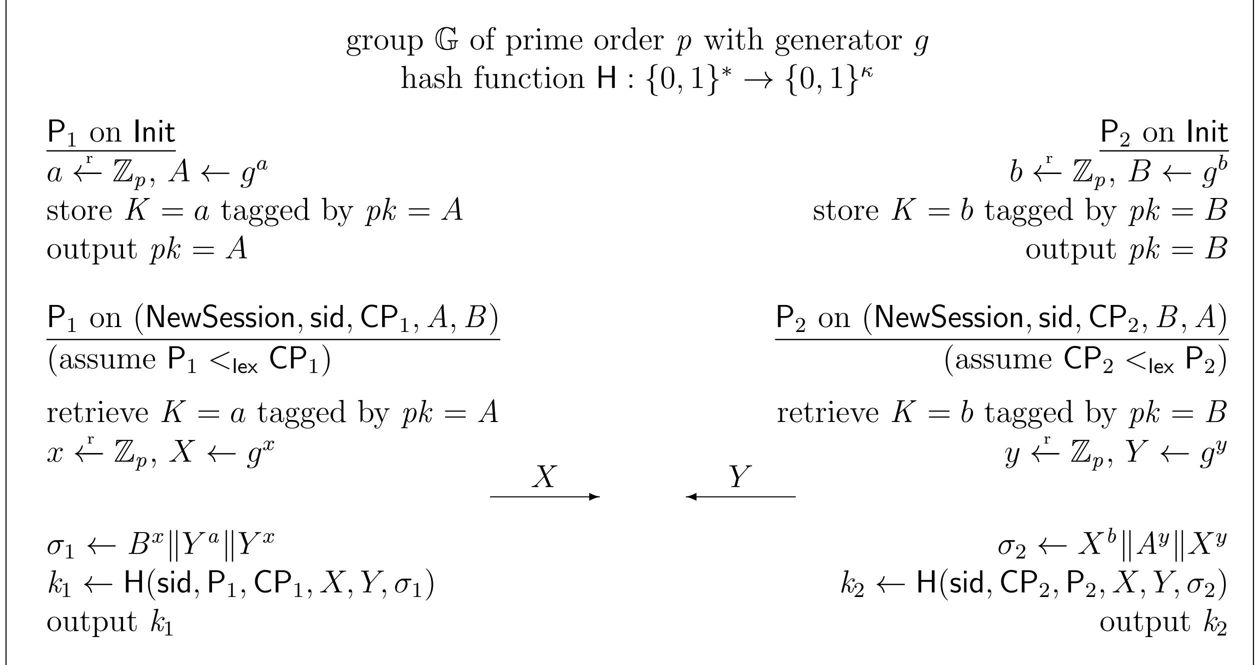


Figure 3.2: Protocol $3DH$: "Triple Diffie-Hellman" Key Exchange

Conventions.

(1) In Figure 3.2 we assume that each party runs 3DH using key pair (K, pk) previously generated via procedure Init. In Figure 3.2 these are resp. (a, A) for P_1 and (b, B) for P_2. Note that no such requirement is posed on the counterparty public key each party uses, resp. public key B used by P_1 and A used by P_2.

(2) We implicitly assume that each party P_i uses its own identity as a protocol input, together with the identity CP_i of its assumed counterparty. These identities could be e.g. domain names, user names, or any other identifiers. They have no other semantics except that the two parties can establish the same session key only if they assume matching identifiers, i.e.

$(\mathsf{P}_1, \mathsf{CP}_1) = (\mathsf{CP}_2, \mathsf{P}_2)$.

(3) Protocol 3DH is symmetric except for the ordering of group elements in tuple σ and the ordering of elements in the inputs to hash H. Each protocol party P can locally determine this order based on whether string P is lexicographically smaller than string CP. (In Figure 3.2 we assume that $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{P}_2$.) An equivalent way to see it is that each party P computes a "role" bit $\mathsf{role} \in \{1, 2\}$ and follows the protocol of party $\mathsf{P}_{\mathsf{role}}$ in Figure 3.2: Party P sets this bit as $\mathsf{role} = 1$, called the "client role", if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$, and $\mathsf{role} = 2$, called the "server role", otherwise.

(4) We assume that parties verify public keys and ephemeral DH values, resp. B, Y for P_1 and A, X for P_2, as group $\mathbb{G}$ elements. Optionally, instead of group membership testing one can use cofactor exponentiation to compute σ.

Theorem 3.1. *Protocol 3DH shown in Figure 3.2 realizes $\mathcal{F}_{\mathsf{khAKE}}$ if the Gap CDH assumption holds and H is a random oracle.*

Proof Overview. We show that that for any efficient environment algorithm $\mathcal{Z}$, its view of the *real-world* security game, i.e. an interaction between the real-world adversary and honest parties who follow protocol 3DH, is indistinguishable from its view of the *ideal-world* game, i.e. an interaction between the ideal-world adversary, whose role is played by the *simulator*, with the functionality $\mathcal{F}_{\mathsf{khAKE}}$. We show the simulator algorithm SIM in Figure 3.3. The real-world game, Game 0, is shown in Figure 3.4, and the ideal-world game defined by a composition of algorithm SIM and functionality $\mathcal{F}_{\mathsf{khAKE}}$, denoted Game 7, is shown in Figure 3.5.

As is standard, we assume that the real-world adversary $\mathcal{A}$ is a subroutine of the environment $\mathcal{Z}$, therefore the sole party that interacts with Games 0 or 7 is $\mathcal{Z}$, issuing commands Init and $\mathsf{NewSession}$ to honest parties P, adaptively compromising public keys, and using $\mathcal{A}$ to

Initialization: Initialize an empty list KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (K, pk) \in KL_{\mathsf{P}}$ send K to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \xleftarrow{r} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w \rangle$, send $W = g^w$ to $\mathcal{A}$

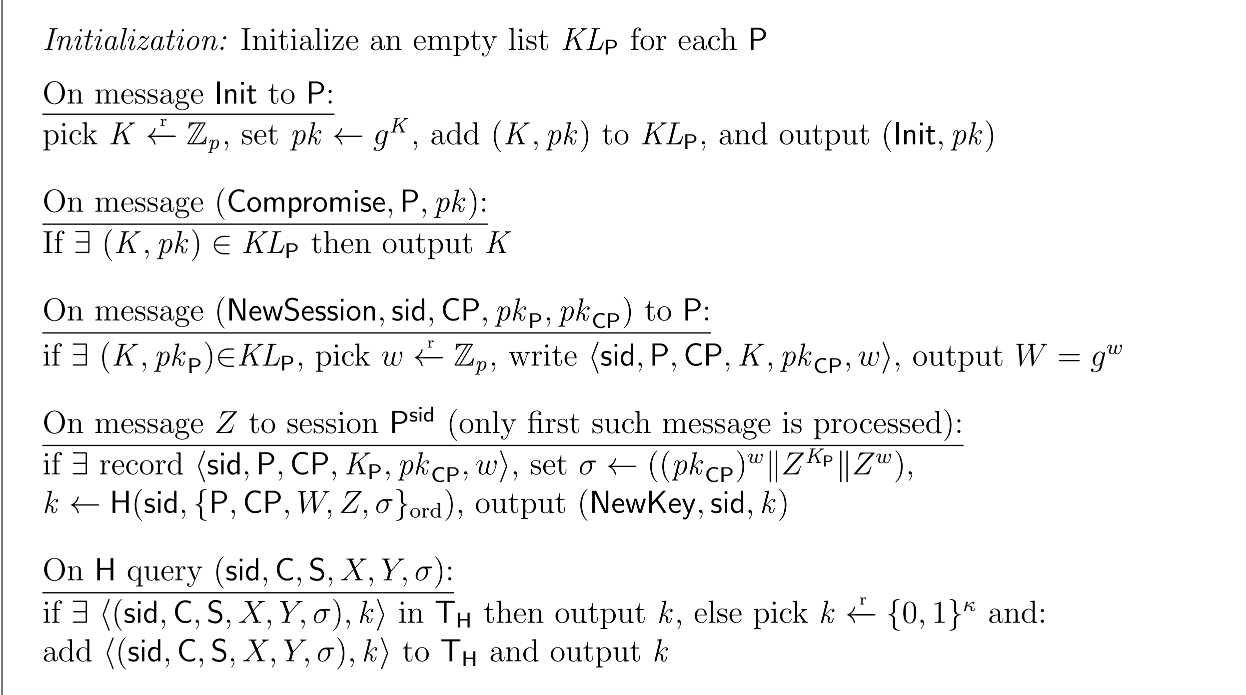
On $\mathcal{A}$'s message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w \rangle$:
    if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z \rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
    send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, Z)$ to $\mathcal{F}$

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ to random oracle H:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t. $(X, \sigma) = (g^x, (B^x \| Y^a \| Y^x))$ for some B,
send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t. $(Y, \sigma) = (g^y, (X^b \| A^y \| X^y))$ for some A,
send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 3.3: Simulator SIM showing that 3DH realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

send protocol messages Z to honest party's sesssions and making hash function H queries. The proof follows a standard strategy of showing a sequence of games that bridge between Game 0 and Game 7, where at each transition we argue that the change is indistinguishable. We use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if we show that $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G7}]|$ is negligible under the stated assumptions.

Notation. To make the real-world interaction in Figure 3.4 more concise, we adopt a notation which stresses the symmetric nature of 3DH protocol: We use variable $W = g^w$ to denote the message which party P sends out, and variable Z to denote the message it receives, e.g. $(W, Z) = (X, Y)$ if P plays the "client" role and $(W, Z) = (Y, X)$ if P plays the "server" role. If $\sigma = \sigma_1 \| \sigma_2 \| \sigma_3$ then let $\{\sigma\}_{\text{flip}} = \sigma_2 \| \sigma_1 \| \sigma_3$. We will use $\{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}}$ to denote string $\mathsf{P}, \mathsf{CP}, W, Z, \sigma$ if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ or string $\mathsf{CP}, \mathsf{P}, Z, W, \{\sigma\}_{\text{flip}}$ if $\mathsf{CP} <_{\mathsf{lex}} \mathsf{P}$. With



Initialization: Initialize an empty list KL_{P} for each P

On message Init to P:
pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$, pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K, pk_{\mathsf{CP}}, w \rangle$, output $W = g^w$

On message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$, set $\sigma \leftarrow ((pk_{\mathsf{CP}})^w \| Z^{K_{\mathsf{P}}} \| Z^w)$,
$k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in T_{H} then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to T_{H} and output k

Figure 3.4: 3DH: Environment's view of real-world interaction (Game 0)

this notation each party's 3DH protocol code can be restated in the symmetric way, as in Figure 3.4, because session key computation of party P can be denoted in a uniform way as $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$ for $\sigma = (pk_{\mathsf{CP}})^w \| Z^{K_{\mathsf{P}}} \| Z^w$.

We use the same symmetric notation to describe simulator SIM in Figure 3.3 and the ideal-world game implied by SIM and $\mathcal{F}_{\mathsf{khAKE}}$ in Figure 3.5, except for the way SIM treats H oracle queries, which we separate into two cases based on the roles played by the two parties whose sessions are potentially involved in any H query. In H-handling code of SIM we denote the identifiers of the two parties involved in a query as C and S, for the parties playing respectively the client and server roles, and the code that follows uses role-specific notation to handle attacks on the sessions executed respectively by C and S.

Throughout the proof we use $\mathsf{P}^{\mathsf{sid}}$ to denote a session of party P with identifier sid. We use $v_{\mathsf{P}}^{\mathsf{sid}}$ to denote a local variable v pertaining to session $\mathsf{P}^{\mathsf{sid}}$ or a message v which this session receives, and whenever identifier sid is clear from the context we write v_{P} instead of $v_{\mathsf{P}}^{\mathsf{sid}}$. Note that session $\mathsf{CP}^{\mathsf{sid}}$ is uniquely defined for every session $\mathsf{P}^{\mathsf{sid}}$ by setting $\mathsf{CP} = \mathsf{CP}_{\mathsf{P}}^{\mathsf{sid}}$, and

we will implicitly assume below that a counterparty's session is defined in this way.

For a fixed environment $\mathcal{Z}$, let q_{K} and q_{ses} be (the upper-bounds on) the number of resp. keys and sessions initialized by $\mathcal{Z}$, let q_{H} be the number of H oracle queries $\mathcal{Z}$ makes, and let $\epsilon^{\mathcal{Z}}_{\text{g-cdh}}$ be the maximum advantage in solving Gap CDH in $\mathbb{G}$ of an algorithm that makes q_{H} DDH oracle queries and uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{H}} + q_{\text{ses}})$ exponentiations in $\mathbb{G}$.

Define the following two functions for every session $\mathsf{P}^{\mathsf{sid}}$:

$$\mathsf{3DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z) = \mathsf{cdh}_g(W, pk') \| \mathsf{cdh}_g(pk, Z) \| \mathsf{cdh}_g(W, Z) \text{ for } W = W^{\mathsf{sid}}_{\mathsf{P}} \quad (3.1)$$

$$R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z) = \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}^{\mathsf{sid}}_{\mathsf{P}}, W^{\mathsf{sid}}_{\mathsf{P}}, Z, \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)\}_{\text{ord}}) \quad (3.2)$$

If session $\mathsf{P}^{\mathsf{sid}}$ runs on its own private key K_{P}, counterparty's public key pk_{CP}, and receives message Z, then its output session key is $k = R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for $pk_{\mathsf{P}} = g^{K_{\mathsf{P}}}$. Note also that an adversary can locally compute function $R^{\mathsf{sid}}_{\mathsf{P}}$ for any pk_{P}, any key pk_{CP} which was either generated by the adversary or it was generated by an honest party but it has been compromised, and any Z which the adversary generates, because the adversary can then compute functions $\mathsf{cdh}_g(\cdot, pk_{\mathsf{CP}})$ and $\mathsf{cdh}_g(\cdot, Z)$ on any inputs.

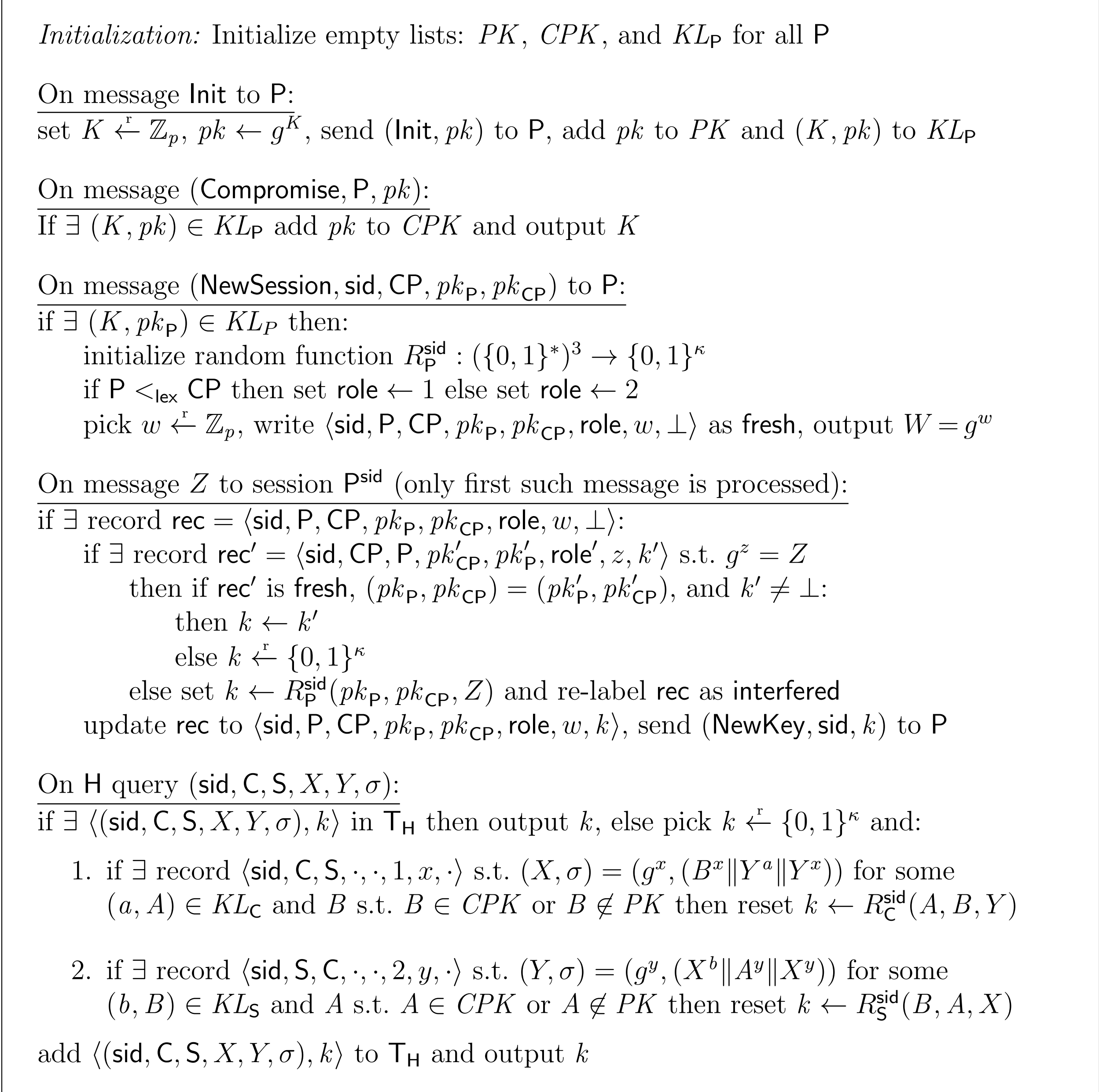
Simulator. Simulator SIM, shown in Figure 3.3, picks all (K, pk) pairs on behalf of honest players and surrenders the corresponding private key whenever an honestly-generated public key is compromised. To simulate honest party P behavior the simulator sends $W = g^w$ for random w. When $\mathsf{P}^{\mathsf{sid}}$ receives Z the simulator forks: If Z originated from honest session $\mathsf{CP}^{\mathsf{sid}}$ which runs on matching identifiers $(\mathsf{sid}, \mathsf{CP}, \mathsf{P})$, SIM treats this as a case of honest-but-curious attack that connects two potentially matching sessions and sends NewKey to $\mathcal{F}_{\mathsf{khAKE}}$. ($Z$ included in this call is ignored by $\mathcal{F}_{\mathsf{khAKE}}$.) Otherwise SIM treats it as an active attack on $\mathsf{P}^{\mathsf{sid}}$ and sends $\mathsf{Interfere}$ followed by $(\mathsf{NewKey}, ..., Z)$. Note that in response $\mathcal{F}_{\mathsf{khAKE}}$ will treat

$\mathsf{P}^{\mathsf{sid}}$ as interfered and set its output key as $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the (own,counterparty) pair of public keys which $\mathsf{P}^{\mathsf{sid}}$ uses, and which is unknown to SIM (except if pk_{CP} was generated by the adversary, in which case it was leaked to SIM at NewSession). Finally, SIM services H oracle queries $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ by identifying those that pertain to viable session-key computations by either session $\mathsf{C}^{\mathsf{sid}}$ or $\mathsf{S}^{\mathsf{sid}}$. We describe it here only for $\mathsf{C}^{\mathsf{sid}}$-side H queries since $\mathsf{S}^{\mathsf{sid}}$-side queries are handled symmetrically. If H query involves $\sigma = \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$ for some A, B s.t. (1) A is one of the public keys generated by C, and (2) B is either some compromised honestly generated public key or it is an adversarial key which $\mathsf{C}^{\mathsf{sid}}$ uses for the counterparty (recall that if $\mathsf{C}^{\mathsf{sid}}$ runs on an adversary-generated counterparty key pk_{CP} then functionality $\mathcal{F}_{\mathsf{khAKE}}$ leaks it to the adversary), then SIM treats that query as a potential computation of a session key output by $\mathsf{C}^{\mathsf{sid}}$, queries $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}_{\mathsf{khAKE}}$. If B is compromised or adverarial then $\mathcal{F}_{\mathsf{khAKE}}$ responds with $k^* \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$ and SIM embeds k^* into H output. Note that if (A, B) matches the (own,counterparty) keys used by $\mathsf{C}^{\mathsf{sid}}$, and $\mathsf{C}^{\mathsf{sid}}$ receives $Z = Y$ in the protocol, then k^* will match the session key output by $\mathsf{C}^{\mathsf{sid}}$. For all other triples (A, B, Y) the outputs of $R^{\mathsf{sid}}_{\mathsf{C}}$ are irrelevant except that (1) if the adversary learns the real session key output by $\mathsf{C}^{\mathsf{sid}}$ then these H outputs inform the adversary that pair (A, B) is *not* the (own,counterparty) key pair used by $\mathsf{C}^{\mathsf{sid}}$, and (2) if the adversary bets on some (A, B) pair used by $\mathsf{C}^{\mathsf{sid}}$ then it can use H queries to find an "optimal" protocol response Y to $\mathsf{C}^{\mathsf{sid}}$ for which the resulting (randomly sampled) session key has some properties the adversary likes, e.g. its last 20 bits are all zeroes, etc.

Game Sequence from Game 0 to Game 7.

Game 0 *(real world)*: This is the interaction of environment $\mathcal{Z}$ (and its subroutine, the real-world adversary) with protocol 3DH, as shown in Fig. 3.4.

Game 1 *(past* H *queries are irrelevent to new sessions)*: Game 1 adds an abort if NewSession



Initialization: Initialize empty lists: PK, CPK, and KL_{P} for all P

On message Init to P:
set $K \xleftarrow{r} \mathbb{Z}_p$, $pk \leftarrow g^K$, send (Init, pk) to P, add pk to PK and (K, pk) to KL_{P}

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ add pk to CPK and output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_P$ then:
    initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$
    if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
    pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$ as fresh, output $W = g^w$

On message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$:
    if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \mathsf{role}', z, k' \rangle$ s.t. $g^z = Z$
        then if rec' is fresh, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$, and $k' \neq \perp$:
            then $k \leftarrow k'$
            else $k \xleftarrow{r} \{0,1\}^\kappa$
        else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ and re-label rec as $\mathsf{interfered}$
    update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k \rangle$, send $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot \rangle$ s.t. $(X, \sigma) = (g^x, (B^x \| Y^a \| Y^x))$ for some $(a, A) \in KL_{\mathsf{C}}$ and B s.t. $B \in CPK$ or $B \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$

2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, \cdot \rangle$ s.t. $(Y, \sigma) = (g^y, (X^b \| A^y \| X^y))$ for some $(b, B) \in KL_{\mathsf{S}}$ and A s.t. $A \in CPK$ or $A \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$

add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 3.5: 3DH: Environment's view of ideal-world interaction (Game 7)

initializes session $\mathsf{P}^{\mathsf{sid}}$ with $W = g^w$ s.t. H has been queried on any tuple of the form $(\mathsf{sid}, \{\mathsf{P}, \cdot, W, \cdot, \cdot\}_{\text{ord}})$. Since each H query can pertain to at most two sessions, $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$, there at most q_{H} such queries, and $w \xleftarrow{r} \mathbb{Z}_p$, we have:

$$|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$$

Game 2 *(programming $R_{\mathsf{P}}^{\mathsf{sid}}$ values into H outputs)*: Define sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ to be *matching* if $\mathsf{CP}_{\mathsf{C}}^{\mathsf{sid}} = \mathsf{S}$ and $\mathsf{CP}_{\mathsf{S}}^{\mathsf{sid}} = \mathsf{C}$. Note that for any matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ and any public keys A, B correctness of 3DH implies that $R_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y_{\mathsf{S}}) = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, X_{\mathsf{C}})$. While in equation (3.2) we defined function $R_{\mathsf{P}}^{\mathsf{sid}}$ in terms of hash H, in Game 2 we set H outputs using appropriately chosen functions $R_{\mathsf{P}}^{\mathsf{sid}}$. For every pair of matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ consider a pair of random functions $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}} : (\mathbb{G})^3 \to \{0,1\}^\kappa$ s.t.

$$R_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y_{\mathsf{S}}^{\mathsf{sid}}) = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, X_{\mathsf{C}}^{\mathsf{sid}}) \quad \text{for all } A, B \in \mathbb{G} \tag{3.3}$$

More precisely, for any session $\mathsf{P}^{\mathsf{sid}}$ with no matching session $R_{\mathsf{P}}^{\mathsf{sid}}$ is set as a random function, and for $\mathsf{P}^{\mathsf{sid}}$ for which a prior matching session exists $R_{\mathsf{P}}^{\mathsf{sid}}$ is set as a random function subject to constraint (4.5). Let PK be the list of all public keys generated so far, and PK_{P} be the set of keys generated for P. Let $PK^+(\mathsf{P}^{\mathsf{sid}})$ stand for $PK \cup \{pk_{\mathsf{CP}}\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$. (If $pk_{\mathsf{CP}} \in PK$ then $PK^+(\mathsf{P}^{\mathsf{sid}}) = PK$.) Consider an oracle H which responds to each new query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $\mathsf{C} <_{\mathsf{lex}} \mathsf{S}$ as follows:

1. If $\exists\ \mathsf{C}^{\mathsf{sid}}$ s.t. $(\mathsf{S}, X) = (\mathsf{CP}_{\mathsf{C}}^{\mathsf{sid}}, X_{\mathsf{C}}^{\mathsf{sid}})$, and $\exists\ A, B$ s.t. $A \in PK_{\mathsf{C}}$, $B \in PK^+(\mathsf{C}^{\mathsf{sid}})$, and $\mathsf{3DH}_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y) = \sigma$, then set $k \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y)$
2. If $\exists\ \mathsf{S}^{\mathsf{sid}}$ s.t. $(\mathsf{C}, Y) = (\mathsf{CP}_{\mathsf{S}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\ B, A$ s.t. $B \in PK_{\mathsf{S}}$, $A \in PK^+(\mathsf{S}^{\mathsf{sid}})$, and $\mathsf{3DH}_{\mathsf{S}}^{\mathsf{sid}}(B, A, X) = \{\sigma\}_{\text{flip}}$, then set $k \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, X)$

3. In any other case sample $k \xleftarrow{r} \{0,1\}^\kappa$

Since the game knows each key pair $(K_\mathsf{P}, pk_\mathsf{P})$ generated for each P, and the ephemeral state w of each session P^sid, it can decide for any Z, pk' if $\sigma = \mathsf{3DH}^\mathsf{sid}_\mathsf{P}(pk_\mathsf{P}, pk', Z) = (pk')^w \| Z^{K_\mathsf{P}} \| Z^w$. Note that each value of $R^\mathsf{sid}_\mathsf{P}$ is used to program H on at most one query. Also, if H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ satisfies both conditions then $(X, Y) = (X^\mathsf{sid}_\mathsf{C}, Y^\mathsf{sid}_\mathsf{S}) = (g^x, g^y)$ and $\exists$ A', B', a, b s.t.

$$\mathsf{3DH}^\mathsf{sid}_\mathsf{C}(g^a, B', Y) = (B')^x \| Y^a \| Y^x = X^b \| (A')^y \| X^y = \{\mathsf{3DH}^\mathsf{sid}_\mathsf{S}(g^b, A', X)\}_\text{flip}$$

Since these equations imply that $(A', B') = (g^a, g^b)$, and by equation (4.5), $R^\mathsf{sid}_\mathsf{C}(A', B', Y^\mathsf{sid}_\mathsf{S}) = R^\mathsf{sid}_\mathsf{S}(B', A', X^\mathsf{sid}_\mathsf{C})$, it follows that if both conditions are satisfied then both will program H output to the same value. Thus we conclude:

$$\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]$$

GAME 3 *(direct programming of session keys using random functions $R^\mathsf{sid}_\mathsf{P}$)*: In Game 3 we make the following changes: We mark each initialized session P^sid as fresh, and when $\mathcal{A}$ sends Z to P^sid then we re-label P^sid as $\mathsf{interfered}$ if Z does not equal to the message sent by the matching session CP^sid, i.e. if $Z^\mathsf{sid}_\mathsf{P} \neq W^\mathsf{sid}_\mathsf{CP}$. Secondly, if session P^sid runs on its own key pair $(K_\mathsf{P}, pk_\mathsf{P})$ and intended counterparty public key pk_CP, we say that it runs "under keys $(pk_\mathsf{P}, pk_\mathsf{CP})$". Using this book-keeping, Game 3 modifies session-key computation for session P^sid which runs under keys $(pk_\mathsf{P}, pk_\mathsf{CP})$ as follows:

1. If $k^\mathsf{sid}_\mathsf{CP} \neq \bot$, sessions $\mathsf{P}^\mathsf{sid}, \mathsf{CP}^\mathsf{sid}$ are fresh and *matching*, and CP^sid runs under keys $(pk_\mathsf{CP}, pk_\mathsf{P})$, then $k^\mathsf{sid}_\mathsf{P} \leftarrow k^\mathsf{sid}_\mathsf{CP}$

2. In any other case set $k^\mathsf{sid}_\mathsf{P} \leftarrow R^\mathsf{sid}_\mathsf{P}(pk_\mathsf{P}, pk_\mathsf{CP}, Z)$.

We argue that this change makes no difference to the environment. In Game 2 the session key $k_{\mathsf{P}}^{\mathsf{sid}}$ is computed as $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$ for $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$. However, H on such input is programmed in Game 2 to output $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ if $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for any $pk_{\mathsf{CP}} \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. Since pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ must be in set $PK^{+}(\mathsf{P}^{\mathsf{sid}})$, setting $k_{\mathsf{P}}^{\mathsf{sid}}$ directly as $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ only short-circuits this process. Moreover, since $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ are correlated by equation (4.5), setting $k_{\mathsf{C}}^{\mathsf{sid}}$ as $k_{\mathsf{S}}^{\mathsf{sid}}$ or vice versa, in the case both are fresh, i.e. $Z_{\mathsf{C}}^{\mathsf{sid}} = Y_{\mathsf{S}}^{\mathsf{sid}}$ and $Z_{\mathsf{S}}^{\mathsf{sid}} = X_{\mathsf{C}}^{\mathsf{sid}}$, and sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ run under matching keys, resp. $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (A, B)$ and $(pk_{\mathsf{CP}}, pk_{\mathsf{P}}) = (B, A)$, also does not change the game. Thus we conclude:

$$\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$$

Game 4 *(abort on session-key derivation* H *query for passive sessions)*: We add an abort if oracle H triggers evaluation of $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ for any pk, pk' and $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$. Note that if $\mathsf{P}^{\mathsf{sid}}$ is passively observed, i.e. it remains fresh then value $W_{\mathsf{CP}}^{\mathsf{sid}}$ either has been delivered to $\mathsf{P}^{\mathsf{sid}}$, i.e. $Z_{\mathsf{P}}^{\mathsf{sid}} = W_{\mathsf{CP}}^{\mathsf{sid}}$, or $\mathsf{P}^{\mathsf{sid}}$ is still waiting for message Z. By the code of oracle H in Game 2 the call to $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', W_{\mathsf{CP}}^{\mathsf{sid}})$ is triggered only if H query $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$ satisfies the following for $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ and $W = W_{\mathsf{P}}^{\mathsf{sid}}$:

$$\sigma \;=\; \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z) \;=\; \mathsf{cdh}_g(W, pk') \parallel \mathsf{cdh}_g(pk, Z) \parallel \mathsf{cdh}_g(W, Z)$$

Hardness of computing such tuple relies on the hardness of computing its last element, i.e. $\mathsf{cdh}_g(W, Z)$, because (W, Z) are Diffie-Hellman KE messages sent by honest sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$. We show that solving Gap CDH can be reduced to causing event Bad, defined as the event that adversary makes such H query. Reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in the messages of simulated parties: If $\mathsf{role} = 1$ then $\mathcal{R}$ sends $X = \bar{X}^s$ for $s \xleftarrow{r} \mathbb{Z}_p$ as the message from $\mathsf{C}^{\mathsf{sid}}$, and if $\mathsf{role} = 2$ then $\mathcal{R}$ sends $Y = \bar{Y}^t$ for $t \xleftarrow{r} \mathbb{Z}_p$ as the message

from $\mathsf{S}^{\mathsf{sid}}$. Finally, $\mathcal{R}$ responds to Init by generating keys $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ as in Game 0.

$\mathcal{R}$ does not know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$ corresponding to messages X, Y, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, but it can use the DDH oracle to emulate the way Game 3 services H queries: To test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $X = \bar{X}^s$ satisfies $\sigma = (L\|M\|N) = (pk^x\|Y^a\|Y^x)$ for $x = s \cdot \bar{x}$ and any a, pk, reduction $\mathcal{R}$ checks if $L = \mathsf{cdh}_g(\bar{X}, pk^s)$, $M = Y^a$, and $N = \mathsf{cdh}_g(\bar{X}, Y^s)$. Symmetrically, $\mathcal{R}$ tests if (X, Y, σ) for $Y = \bar{Y}^t$ satisfies $\sigma = (M\|L\|N) = (X^b\|pk^y\|X^y)$ by checking if $L = \mathsf{cdh}_g(\bar{Y}, pk^t)$, $M = X^b$, and $N = \mathsf{cdh}_g(\bar{Y}, X^t)$.

Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3. If it does then $\mathcal{R}$ detects it by checking if the last element N in σ satisfies $N = \mathsf{cdh}_g(W, Z)$ for $W = W_{\mathsf{P}}^{\mathsf{sid}}$ and $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$. If $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ are matching then one of them plays the client role and the other the server role, i.e. either (W, Z) or (Z, W) is equal to $(\bar{X}^s, \bar{Y}^t)$ for some s, t known by $\mathcal{R}$. In either case $\mathcal{R}$ can output $N^{1/(st)}$ as the answer $\mathsf{cdh}_g(\bar{X}, \bar{Y})$ to its CDH challenge. It follows that $\Pr[\mathsf{Bad}] \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$, hence:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Game 5 *(random keys on passively observed sessions)*: We modify the game so that if session $\mathsf{P}^{\mathsf{sid}}$ remains fresh when $\mathcal{A}$ sends Z to $\mathsf{P}^{\mathsf{sid}}$ then instead of setting $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ as in Game 3, we now set $k_{\mathsf{P}}^{\mathsf{sid}} \xleftarrow{r} \{0,1\}^{\kappa}$. Since session $\mathsf{P}^{\mathsf{sid}}$ can remain fresh only if Z it receives was sent by its matching session, i.e. $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$, and by Game 4 oracle H never queries $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for such Z, it follows by randomness of $R_{\mathsf{P}}^{\mathsf{sid}}$ that the modified game remains externally identical, hence:

$$\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$$

GAME 6 *(decorrelating function pairs $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}}$)*: Let Game 6 be as Game 5, except that functions $R^{\mathsf{sid}}_{\mathsf{S}}, R^{\mathsf{sid}}_{\mathsf{S}}$ are chosen without the constraint imposed by equation (4.5). Since by Game 5 neither function is queried on the points which create the correlation imposed by equation (4.5), it follows that:

$$\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$$

GAME 7 *(hash computation consistent only for compromised keys)*: Recall that in Game 6, as in Game 2, $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W^{\mathsf{sid}}_{\mathsf{P}}, Z, \sigma\}_{\text{ord}})$ is defined as $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ if $\sigma = \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ for some $pk \in PK_{\mathsf{P}}$, and $pk' \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. In Game 7 we add a condition that this programming of H can occur only if either (1) pk' is an honestly generated key of some party, but it has been $\mathsf{compromised}$ or (2) pk' is the counterparty key which session $\mathsf{P}^{\mathsf{sid}}$ runs under, and it is an *adverarial* key, i.e. it has not been generated by Init. Note that these are the two cases in which the adversary can know the secret key corresponding to pk', and we will show that this knowledge is indeed necessary for adversary to compute σ s.t. $\sigma = \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$.

Let CPK be the list of generated public keys who were compromised so far, and $CPK^{+}(\mathsf{P}^{\mathsf{sid}})$ stand for CPK if the counterparty public key pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ is an honestly generated key, and for $CPK \cup \{pk_{\mathsf{CP}}\}$ if pk_{CP} is adversarially-generated. The modification of Game 7 is that H output is programmed to $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ for pk' s.t. $\sigma = \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ only if $pk' \in CPK^{+}(\mathsf{P}^{\mathsf{sid}})$, while in Game 6, as in Game 2, this programming was done whenever $pk' \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. Therefore the two games diverge in the case of event Bad defined as H query as above for $pk' \in PK \setminus CPK$, i.e. honestly generated and *not* compromised key. Let Bad_n be Bad where $\mathsf{P}^{\mathsf{sid}}$ plays role $= n$. We show a reduction $\mathcal{R}$ that solves Gap CDH if Bad_1 occurs. The argument for event Bad_2 is symmetrical.

Note that Bad_1 corresponds to H query on string $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $\sigma = (B^x \| Y^a \| Y^x)$

where $x = x_{\mathsf{C}}^{\mathsf{sid}}$, a is some private key of C, and B is a non-compromised public key in PK (not necessarily owned by S). On input a CDH challenge $(\bar{X}, \bar{B})$, $\mathcal{R}$ sets each $X_{\mathsf{C}}^{\mathsf{sid}}$ as $\bar{X}^s$ for random s, just like the reduction in Game 4, but it sets each $Y_{\mathsf{S}}^{\mathsf{sid}}$ as g^y for random y. $\mathcal{R}$ also picks all keys $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ as in Game 0, except for a chosen index $i \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ sets the key generated in the i-th call to Init (by any party P) as $pk[i] \leftarrow \bar{B}$. Let $\mathsf{Bad}_1[i]$ denote event Bad_1 occuring for B which is this i-th key, i.e. $B = \bar{B}$.

As long as key $pk[i]$ is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys, and it can service H queries as follows: To test server-side σ's, i.e. if $\sigma = (M \| L \| N) = (X^K \| pk^y \| X^y)$, reduction $\mathcal{R}$ tests it as Game 6 does except for K that corresponds to the public key $\bar{B}$, in which case it tests if $M = \mathsf{cdh}_g(\bar{B}, X)$, $L = pk^y$, and $N = X^y$. To test client-side σ's, i.e. if $\sigma = (L \| M \| N) = (pk^x \| Y^K \| Y^x)$ for $x = s \cdot \bar{x}$ where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and any pk, including $pk = \bar{B}$, reduction $\mathcal{R}$ tests if $L = \mathsf{cdh}_g(\bar{X}, pk^s)$, $M = Y^K$, and $N = \mathsf{cdh}_g(\bar{X}, Y^s)$, except for the case that K is the private key corresponding to the public key $\bar{B}$, in which case $\mathcal{R}$ replaces test $M = Y^K$ with $M = \mathsf{cdh}_g(\bar{B}, Y)$.

Note that $\mathsf{Bad}_1[i]$ can happen only before key $pk[i]$ is compromised, so event $\mathsf{Bad}_1[i]$ occurs in the reduction with the same probability as in Game 6. (If $\mathcal{A}$ asks to compromise of $pk[i]$ then $\mathcal{R}$ aborts.) $\mathcal{R}$ can detect event $\mathsf{Bad}_1[i]$ because it occurs if H query involves the public key $pk[i] = \bar{B}$ and σ satisfies the client-side equation for this key, in which case $\mathcal{R}$ can output $L^{1/s} = \mathsf{cdh}_g(\bar{X}, \bar{B})$. If $\mathcal{R}$ picks index i at random it follows that $\Pr[\mathsf{Bad}_1] \leq q_{\mathrm{K}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$. Since a symmetric argument holds also for $\Pr[\mathsf{Bad}_2]$, we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_{\mathrm{K}}) \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Observe that Game 7 is identical to the ideal-world game shown in Figure 3.4: By Game 6 all functions $R_{\mathsf{P}}^{\mathsf{sid}}$ are random, by Game 5 the game responds to Z messages to $\mathsf{P}^{\mathsf{sid}}$ as the game in Figure 3.4, and after the modification in oracle H done in Game 7 this oracle also acts as in

Figure 3.4. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 3.1.

3.4 HMQV as Key-Hiding AKE

We show that protocol HMQV [100], presented in Figure 3.6, realizes the UC notion of Key-Hiding AKE, as defined by functionality $\mathcal{F}_{\mathsf{khAKE}}$ in Section 4.2, under the Gap CDH assumption in ROM. It allows us to use HMQV with KHAPE, resulting in its most efficient instantiation, and, to the best of our knowledge the most efficient aPAKE protocol proposed. HMQV has been analyzed in [100] under the game-based AKE model of Canetti and Krawczyk [49], but the analysis we present is the first, to the best of our knowledge, to be done in the UC model.[6]

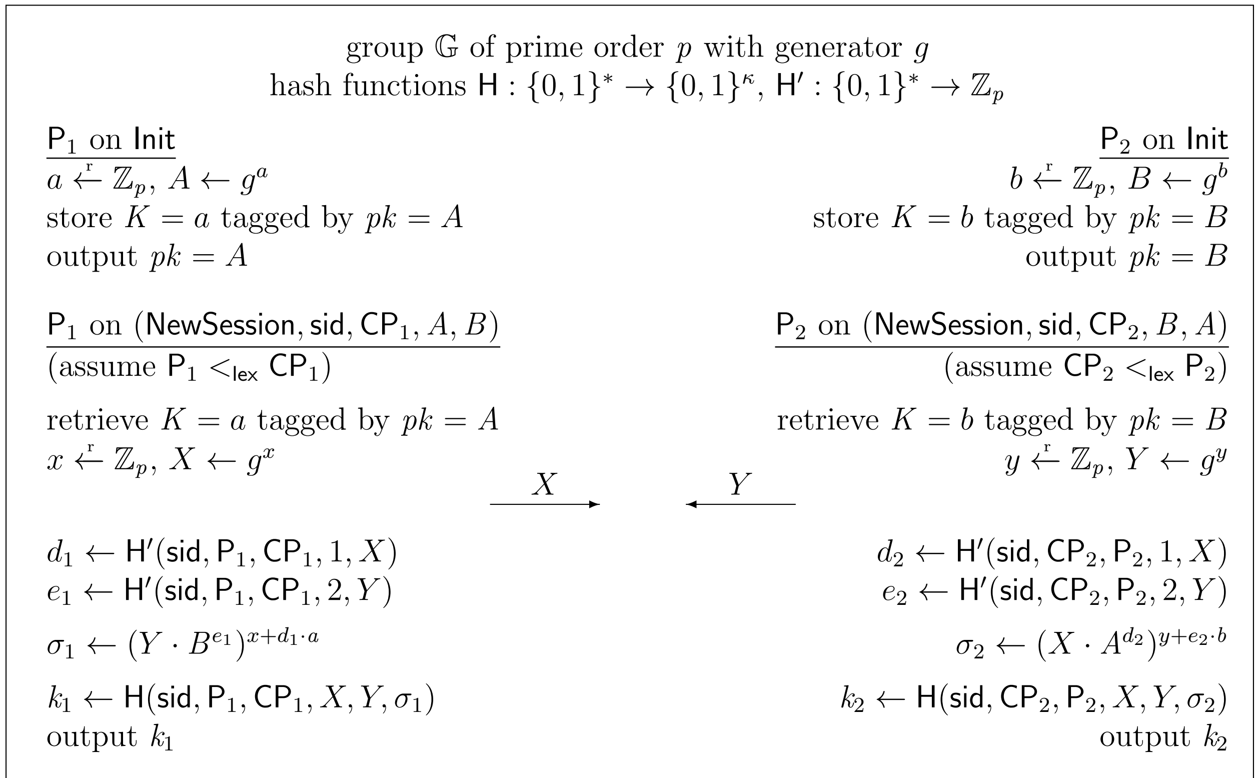


Figure 3.6: Protocol HMQV [100]

[6] However, we do not include adaptive session state compromise considered in [49, 100].

The logic of why HMQV is key hiding is similar to the case of 3DH. Namely, the only way to attack the privacy of party P which runs HMQV on inputs $(K, pk) = (a, B)$, is to compromise the private key b corresponding to the public key B. (And symmetrically for the party that runs on $(K, pk) = (b, A)$.) The HMQV equation, just like the 3DH key equation, involves both the ephemeral sessions secrets (x, y) and the long-term keys (a, b), combining them in a DH-like formula $\sigma = g^{(x+da)\cdot(y+eb)}$ where d, e are hashes of (session state identifiers and) resp. $X = g^x$ and $Y = g^y$. Following essentially the same arithmetics as in the proof due to [100] shows that the only way to compute σ is to know either *both* x, a or *both* y, b, which means that the attacker must be (1) active, to chose the ephemeral session state variable resp. x or y, and (2) it must know the counterparty private key, resp. a or b.

Theorem 3.2. *Protocol HMQV shown in Figure 3.6 realizes $\mathcal{F}_{\mathsf{khAKE}}$ if the Gap CDH assumption holds and H, H' are random oracles.*

The proof of theorem 3.2 follows the template of the proof for the corresponding theorem on 3DH security, i.e. theorem 3.1, and we show the proof below:

Proof. We describe how the security proof for 3DH should be adapted to the case of HMQV. The two proofs follow the same template. In particular, the HMQV simulator algorithm SIM, shown in Figure 3.7, acts very similarly to the 3DH simulator in Figure 3.3. The proof shows the indistinguishability between the real-world game (Game 0) shown in Figure 3.8, which captures an interaction with parties running the HMQV protocol, and the ideal-world game (Game 7) shown in Figure 4.10, which is defined by a composition of SIM and functionality $\mathcal{F}_{\mathsf{khAKE}}$. The sequence of games which shows this indistinguishability is exactly the same as the sequence used in the proof of theorem 3.1, and below we sketch where the match is exact and how we deal with the HMQV-specific differences when they occur. In particular, in the discussions below we often re-use the notation introduced used in the proof of theorem 3.1.

As in the case of 3DH, for each AKE session $\mathsf{P}^{\mathsf{sid}}$ we define function $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ which

Initialization: Initialize an empty list KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
pick $K \stackrel{r}{\leftarrow} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (K, pk) \in KL_{\mathsf{P}}$ send K to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \stackrel{r}{\leftarrow} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w \rangle$, send $W = g^w$ to $\mathcal{A}$

On $\mathcal{A}$'s message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w \rangle$:
  if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z \rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
  send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, Z)$ to $\mathcal{F}$

On query (st, σ) to random oracle H, for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y)$:
if $\exists\ \langle (\mathsf{st}, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, otherwise pick $k \stackrel{r}{\leftarrow} \{0,1\}^\kappa$ and:

  if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x \rangle$, $(a, A) \in KL_{\mathsf{C}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(X, \sigma) = (g^x, (Y \cdot B^e)^{x+da})$ for some B:
    send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$

  if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y \rangle$, $(b, B) \in KL_{\mathsf{S}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(Y, \sigma) = (g^y, (X \cdot A^d)^{y+eb})$ for some A:
    send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$

  add $\langle (\mathsf{st}, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$ to random oracle H':
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T_{H'}}$ then output r
else pick $r \stackrel{r}{\leftarrow} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T_{H'}}$, and output r

Figure 3.7: Simulator SIM showing that HMQV realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

is used by session $\mathsf{P}^{\mathsf{sid}}$ to compute its session key given counterparty's message Z. The definition of $R^{\mathsf{sid}}_{\mathsf{P}}$ is exactly the same as in the case of 3DH, i.e. equation (3.2), except the last argument, σ, is now defined using the HMQV function, $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$. Below we define function $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ for session $\mathsf{P}^{\mathsf{sid}}$ running on inputs $(\mathsf{sid}, \mathsf{CP}, pk, pk')$, i.e. pk is its own public key and pk' is the public key of the intended counterparty. Function $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ can be defined separately for cases for $\mathsf{P}^{\mathsf{sid}}$ playing the client-role, denoted $\mathsf{P} = \mathsf{C}$, and $\mathsf{P}^{\mathsf{sid}}$

Initialization: Initialize an empty list KL_{P} for each P

On message Init to P:
pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$; if $\exists\ (K_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$, pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, K_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$, output $W = g^w$

On message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, K_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$:
 if $\mathsf{role} = 1$:
  set $d \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\text{ord}}, 1, g^w)$ and $e \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\text{ord}}, 2, Z)$
  set $\sigma \leftarrow (Z \cdot pk_{\mathsf{CP}}^{e})^{w + d \cdot K_{\mathsf{P}}}$
 if $\mathsf{role} = 2$:
  set $d \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\text{ord}}, 1, Z)$ and $e \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\text{ord}}, 2, g^w)$
  set $\sigma \leftarrow (Z \cdot pk_{\mathsf{CP}}^{d})^{w + e \cdot K_{\mathsf{P}}}$
 set $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k
else pick $k \xleftarrow{r} \{0,1\}^\kappa$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$, and output k

On H' query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T_{H'}}$ then output r
else pick $r \xleftarrow{r} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T_{H'}}$, and output r

Figure 3.8: HMQV: Environment's view of real-world interaction (Game 0)

playing the server-role, denoted $\mathsf{P} = \mathsf{S}$, as follows:

$$\begin{aligned}
\mathsf{HMQV}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', Y) = \quad & \mathsf{cdh}_g(X, Y) \cdot \mathsf{cdh}_g(pk', X)^e \cdot \mathsf{cdh}_g(Y, pk)^d \cdot \mathsf{cdh}_g(pk, pk')^{ed} \\
\text{for } X = X_{\mathsf{C}}^{\mathsf{sid}} \text{ and} \quad & d = \mathsf{H}'(\mathsf{st}, 1, X), e = \mathsf{H}'(\mathsf{st}, 2, Y), \mathsf{st} = \mathsf{sid}|\mathsf{C}|\mathsf{S} \\
\mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', X) = \quad & \mathsf{cdh}_g(X, Y) \cdot \mathsf{cdh}_g(pk, X)^e \cdot \mathsf{cdh}_g(Y, pk')^d \cdot \mathsf{cdh}_g(pk, pk')^{ed} \\
\text{for } Y = Y_{\mathsf{S}}^{\mathsf{sid}} \text{ and} \quad & d = \mathsf{H}'(\mathsf{st}, 1, X), e = \mathsf{H}'(\mathsf{st}, 2, Y), \mathsf{st} = \mathsf{sid}|\mathsf{C}|\mathsf{S}
\end{aligned}$$

Game 0 *(real world)*: The real-world game is shown in Figure 3.8.

Game 1 *(past* H *queries are irrelevent to new sessions)*: We add an abort if session $\mathsf{P}^{\mathsf{sid}}$ starts with W which appeared in some prior inputs to H. As in the case of 3DH, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$.

Game 2 *(programming* $R_{\mathsf{P}}^{\mathsf{sid}}$ *values into* H *outputs)*: We make the same change of using random but pair-wise correlated functions $R_{\mathsf{P}}^{\mathsf{sid}}$, i.e. correlated as in equation (4.5), and programming $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ values into outputs of $\mathsf{H}(\mathsf{sid}, \{\mathsf{C}, \mathsf{S}, W, Z\}_{\mathrm{ord}}, \sigma)$ if W matches the value sent by $\mathsf{P}^{\mathsf{sid}}$ and $\sigma = \mathsf{HMQV}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$. As in the case of 3DH we need to argue that if the same hash query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$, for $(X, Y) = (X_{\mathsf{C}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, matches both the client-side equation and the server-side equation, i.e. if

$$\sigma = \mathsf{HMQV}_{\mathsf{C}}^{\mathsf{sid}}(A, B', Y) = \mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(B, A', X)$$

where $A \in PK_{\mathsf{C}}, B' \in PK^{+}(\mathsf{C}^{\mathsf{sid}}), B \in PK_{\mathsf{S}}, A' \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$, as defined in the 3DH proof, then either condition programs the same value into H output.

In the case of 3DH the corresponding equation implied that both parties must use correct counterparty keys, i.e. that $(A', B') = (A, B)$, in which case constraint (4.5) on $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ implies that either condition programs H to the same value.

In the case of HMQV the above equation can hold even if $(A', B') \neq (A, B)$, but it can occur with only negligible probability. The constraint above implies:

$$(Y \cdot B'^{e})^{x+da} = (X \cdot A'^{d})^{y+eb} \tag{3.4}$$

where $d = \mathsf{H}'(\mathsf{st}, 1, X)$ and $e = \mathsf{H}'(\mathsf{st}, 2, Y)$ and $(X, Y) = (g^x, g^y)$. Note that equation (4.6) holds if and only if $(y + eb')(x + da) = (x + a'd)(y + eb)$, where a', b' are the discrete logarithms

Initialization: Initialize empty lists: PK, CPK, and KL_{P} for all P

On message Init to P:
set $K \xleftarrow{r} \mathbb{Z}_p$, $pk \leftarrow g^K$, send (Init, pk) to P, add pk to PK and (K, pk) to KL_{P}

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ add pk to CPK and output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_P$ then:
  initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$
  if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
  pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$ as fresh, output $W = g^w$

On message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$:
  if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \mathsf{role}', z, k' \rangle$ s.t. $g^z = Z$
    then if rec' is fresh, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$, and $k' \neq \perp$:
      then $k \leftarrow k'$
      else $k \xleftarrow{r} \{0,1\}^\kappa$
    else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ and re-label rec as $\mathsf{interfered}$
  update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k \rangle$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$ and $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(X, \sigma) = (g^x, (Y \cdot B^e)^{x+d\cdot a})$ for some $(a, A) \in KL_{\mathsf{C}}$ and B s.t. $B \in CPK$ or $B \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$

2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, \cdot \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$ and $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(Y, \sigma) = (g^y, (X \cdot A^d)^{y+e\cdot b})$ for some $(b, B) \in KL_{\mathsf{S}}$ and A s.t. $A \in CPK$ or $A \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$

add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

On H' query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T_{H'}}$ then output r
else pick $r \xleftarrow{r} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T_{H'}}$, and output r

Figure 3.9: HMQV: Environment's view of ideal-world interaction (Game 7)

of resp. A, B. This can hold even if $(a', b') \neq (a, b)$, hence in the case of HMQV we will add an abort in the case equation (4.6) holds and $(A', B') \neq (A, B)$. Note that the adversary must choose the counterparty key $pk' = B'$ for session $\mathsf{C}^{\mathsf{sid}}$ before $\mathsf{C}^{\mathsf{sid}}$ starts and picks x. Likewise $pk' = A'$ for session $\mathsf{S}^{\mathsf{sid}}$ must be chosen before $\mathsf{S}^{\mathsf{sid}}$ picks y. Therefore the last value to be chosen is either x or y, i.e. either x or y are randomly sampled after (a, b, a', b') are all fixed. If x is chosen after (a, b, a', b', y) then its choice determines $d = \mathsf{H}'(\mathsf{st}, 1, g^x)$, but since H' is a random oracle, the probability that d satisfies equation (4.6) is $1/p$. Since a symmetric argument holds in the case y (and e) are chosen last, it follows that:

$$|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq q_{\text{ses}}/p$$

GAME 3 *(direct programming of session keys using random functions $R^{\mathsf{sid}}_{\mathsf{P}}$)*: This step is identical as in the case of 3DH, and $\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$

GAME 4 *(abort on H queries for passive sessions)*: As in the case of the proof for 3DH we add an abort whenever oracle H triggers evaluate of $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ for any pk, pk' and $Z = W^{\mathsf{sid}}_{\mathsf{CP}}$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$, and likewise define as Bad the event that such query is made. W.l.o.g. we can consider these sessions using arbitrary pk's, e.g. B' for $\mathsf{C}^{\mathsf{sid}}$ and A' for $\mathsf{S}^{\mathsf{sid}}$, which might or might not equal to the correct public key of the intended counterparty on the respective session.

As in the case of 3DH we show that solving Gap CDH can be reduced to causing event Bad in this game, but the full reduction $\mathcal{R}'$ that exihbits that uses rewinding over two executions of a subsidiary reduction $\mathcal{R}$, which works as follows. We argue reduction $\mathcal{R}$ assuming that event Bad occurs for a client-side function $R^{\mathsf{sid}}_{\mathsf{C}}$, because the case for a server-side function $R^{\mathsf{sid}}_{\mathsf{S}}$ is symmetric.

Reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in a randomized way in the

messages of all simulated parties, i.e. it sends $X = \bar{X}^s$ and $Y = \bar{Y}^t$ for random s and t shifts on behalf of resp. $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ sections, just like in the 3DH case. Otherwise it emulates the security game, in particular it knows all the key pairs (a, A) and (b, B). Although $\mathcal{R}$ does not know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$ corresponding to these messages, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, reduction $\mathcal{R}$ can use the DDH oracle to emulate H queries, i.e. to test if

$$\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{C}}(A, B', Y) = (Y(B')^e)^{x+da} = \mathsf{cdh}_g(X, Y(B')^e) \cdot (Y(B')^e)^{ad}$$

for any key B', any key $A = g^a$ of $\mathsf{C}^{\mathsf{sid}}$, and $(X, Y) = (\bar{X}^s, \bar{Y}^t)$ sent by resp. $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$. Symmetrically $\mathcal{R}$ can test if $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{S}}(B, A', X)$.

Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3, in which case $\mathcal{R}$ can compute $v = \mathsf{cdh}_g(X, Y(B')^e)$, assuming Bad occurs for a client-side equation. Denote this (e, v) pair as (e_1, v_1), i.e. $v_1 = \mathsf{cdh}_g(X, Y(B')^{e_1})$. By the rewinding argument, as in [100], if the probability of the (client-side) Bad is ϵ, the second-layer reduction $\mathcal{R}'$ can run $\mathcal{R}$ against the adversary/environment twice, providing a fresh random output e_2 on hash query $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$. If event Bad occurs in that second execution for the same Y then $\mathcal{R}$ would extract $v_2 = \mathsf{cdh}_g(X, Y(B')^{e_2})$, in which case $\mathcal{R}'$ can compute $\mathsf{cdh}_g(X, Y) = (v_1^{e_2}/v_2^{e_1})^{1/(e_2-e_1)}$, and consequently solve for $\mathsf{cdh}_g(\bar{X}, \bar{Y}) = (\mathsf{cdh}_g(X, Y))^{1/(st)}$. By the standard rewinding argument, the probability $\mathcal{R}'$ succeeds is at least $(1/c_{\mathsf{rwnd}})\epsilon^2/q_{\mathsf{H}}$ for a small constant c_{rwnd}, which implies

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq (c_{\mathsf{rwnd}} \cdot q_{\mathsf{H}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}})^{1/2}$$

Game 5 *(random keys on passively observed sessions)*: This game change is the same as in the case of 3DH, and $\Pr[\mathsf{G5}] = \Pr[\mathsf{G3}]$

Game 6 *(decorrelating function pairs $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}}$)*: This game change is the same as in the

case of 3DH, and $\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$

GAME 7 *(hash computation consistent only for compromised keys)*: As in the proof for 3DH, we restrict handling H queries to only those that correspond to counterparty key pk' being either compromised or adversarial. Consequently, as in the case of 3DH, Game 7 diverges from Game 6 if event Bad occurs, defined as H query on $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W_{\mathsf{P}}^{\mathsf{sid}}, Z\}_{\text{ord}}, \sigma)$ for $\sigma = \mathsf{HMQV}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ where $pk \in PK_{\mathsf{P}}$ and $pk' \in PK \setminus CPK$. Let Bad_n be Bad where $\mathsf{P}^{\mathsf{sid}}$ plays role $= n$. As in the case of the 3DH proof we will argue only for $n = 1$ because the other case is symmetric. Also, we will focus on sub-event $\mathsf{Bad}_1[i]$ which denotes Bad_1 occuring where $\mathsf{P}^{\mathsf{sid}}$ uses the i-th key as pk', i.e. pk' was a non-compromised public key in PK created in the i-th key initialization query. Note that $\mathsf{Bad}_1[i]$ corresponds to $\sigma = (Y \cdot (pk')^e)^{x+d \cdot a}$ where a is some private key of C and x is used on session $\mathsf{C}^{\mathsf{sid}}$, pk' equals to the honestly-generated and non-compromised public key corresponding to the i-th key record, and an arbitrary Y which adversary specifies in the hash inputs.

As in the 3DH proof we show a reduction $\mathcal{R}$ that solves a Gap *Square* DH if $\mathsf{Bad}_1[i]$ occurs. Square DH is a variant of CDH where the challenge is a single value $\bar{X}$ and the goal is to compute $\mathsf{cdh}_g(\bar{X}, \bar{X})$. It is well-known that Square DH is equivalent to CDH. As in the reduction to Gap CDH used in Game 4 above, here too we will use a subsidiary reduction $\mathcal{R}$ which computes CDH on a problem *related* to the Square DH challenge, and a top-level reduction $\mathcal{R}'$ which solves the Square DH challenge using rewinding over two executions of $\mathcal{R}$. We show the bound on the probability that $\mathcal{R}$ succeeds in terms of the probability ϵ of event $\mathsf{Bad}[i]$, and then we show the overall bound using a union bound and a symmetry of client-side and server-side equations.

Reduction $\mathcal{R}$ takes a Square DH challenge $\bar{X}$ and embeds it as $pk' \leftarrow \bar{X}$ where pk' is the public key in the i-th key record, and also embeds $\bar{X}$ into messages $X = \bar{X}^s$ sent on behalf of all client-role sessions, for random s. As in the case of 3DH $\mathcal{R}$ picks all other long-term

key pairs (K, pk) as in the original security game, and it also picks ephemeral state y of all server-side sessions. As long as pk' is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys, and it can service H queries as follows: To test client-side σ's, i.e. if $\sigma = (Y \cdot (pk')^e)^{x+d \cdot K_\mathsf{C}}$ where pk' is an arbitrary public key, Y is an arbitrary value input into the hash, $x = s \cdot \bar{x}$ is an ephemeral state of $\mathsf{C}^{\mathsf{sid}}$ unknown to $\mathcal{R}$, and K_C w.l.o.g. can correspond to the i-th public key $\bar{X}$, hence also unknown to $\mathcal{R}$ (the case of any other key, where $\mathcal{R}$ knows the corresponding key K_C is strictly easier), reduction $\mathcal{R}$ uses the DDH oracle to test if $\sigma = \mathsf{cdh}_g(Y \cdot (pk')^e, \bar{X}^{s+d})$. To test server-side σ's, i.e. if $\sigma = (X \cdot (pk')^d)^{y+e \cdot K_\mathsf{S}}$, for arbitrary X, pk', a known ephemeral state y, and K_S which again w.l.o.g. can correspond to the i-th public key $\bar{X}$, reduction $\mathcal{R}$ uses the DDH orace to test if $\sigma = (X \cdot (pk')^d)^y \cdot \mathsf{cdh}_g(X \cdot (pk')^d, \bar{X}^e)$.

As in the case of 3DH proof this emulation is perfect, so $\mathsf{Bad}_1[i]$ occurs with the same probability as in Game 6, and if does then the client-side equation for σ involves $pk' = \bar{X}$, which means that $\mathcal{R}$ computes $\sigma_1 = \mathsf{cdh}_g(Y \cdot \bar{X}^{e_1}, \bar{X}^{s+d})$, where as in the rewinding reduction in the case of Game 4 above we use e_1 to denote $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$ in the first execution of $\mathcal{R}$. Let $w = \mathsf{cdh}_g(Y, \bar{X})$ and $z = \mathsf{cdh}_g(\bar{X}, \bar{X})$, and note that $\sigma_1 = w^{s+d} \cdot z^{e_1(s+d)}$. If the second run of $\mathcal{R}$ hits the event for the same Y and embeds fresh e_2 into $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$ then it computes $\sigma_2 = w^{s'+d'} \cdot z^{e_2(s'+d')}$. Since $\mathcal{R}'$ knows all the coefficients, it can solve these relations for w, z and output $z = \mathsf{cdh}_g(\bar{X}, \bar{X})$.

If i is randomly chosen and w.l.o.g. Bad_1 is at least as likely as Bad_2 then the probability of $\mathsf{Bad}[i]$ is at least $\epsilon/(2q_\mathrm{K})$. Therefore, by the standard rewinding argument, $\mathcal{R}$ succeeds with probability at least $(1/c_\mathsf{rwnd})(\epsilon/2q_\mathrm{K})^2/q_\mathsf{H}$, which implies

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_\mathrm{K}) \cdot (c_\mathsf{rwnd} \cdot q_\mathsf{H} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}})^{1/2}$$

which concludes the proof. □

3.5 SKEME as Key-Hiding AKE

We present a KEM-based instantiation of the SKEME protocol [97] in Figure 3.10. For compliance with the UC notion of AKE modeled by functionality $\mathcal{F}_{\mathsf{khAKE}}$, we derive the session key via a hash involving several additional elements, including a session identifier sid, party identities C and S, public keys A and B, and the transcript X, c, Y, d. We will also use $\{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\text{ord}}$ to denote $(\mathsf{P}, \mathsf{CP}, A, B, g^w, c, Z, d, (K, L, Z^w))$ if P plays role $= 1$, and string $(\mathsf{CP}, \mathsf{P}, A, B, Z, c, g^w, d, (K, L, Z^w))$ if role $= 2$. Using this notation each party P can derive its session key as $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\text{ord}})$.

The security of the protocol relies on two properties of the underlying KEM. First, we assume KEM to be One-Way under Plaintext-Checking-Attack, abbreviated as OW-PCA[81], where the attacker is given access to a Plaintext-Checking Oracle that on input a key K and ciphertext c, it tells if c decapsulates to K under a given KEM key. See Definition 2.8. Second, we require the KEM to be strong key-anonymous, namely, given two pairs of private-public keys and a key encapsulation under one of them, one cannot distinguish (information-theoretically) which pair generated that ciphertext. Note the correspondence to the notion of key-hiding PKE [24]. The details of strong key-anonymous property is shown in Definition 2.7. However, here to make security reduction easy we use an even stronger version, which we call perfect key-anonymous, i.e. the adversarial advantage of winning strong key-anonymous game is exact 0. The definition is shown below.

Definition 3.1. *[perfect (key-)anonymous] Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}_1, \mathsf{Enc}_2, \mathsf{Dec})$ *be a key-encapsulation mechanism. Let* $b \in \{0, 1\}$*. Let* A *be the adversary. Now we consider the following experiment:*

Experiment $\boldsymbol{Exp}^{perfect-anony-b}_{\mathsf{KEM},A}$

$(pk_0, K_0) \leftarrow \mathsf{KEM.Gen}; (pk_1, K_1) \leftarrow \mathsf{KEM.Gen}$

$c, r \leftarrow \mathsf{KEM.Enc}_1(\kappa),\ K \leftarrow \mathsf{KEM.Enc}_2(pk_b, r)$

$b' \leftarrow A(K_0, pk_0, K_1, pk_1, c)$

Return b'

We also assume KEM we use suffices the security property of One-Wayness under Plaintext-Checking-Attack(OW-PCA)[81], see Definition 2.8. Fortunately many KEMs suffice above two requirements, including plain El Gamal KEM, where the encryption generates $c = g^r$ and $K = pk^r$. And this El Gamal encryption scheme is **OW-PCA** secure based on GapDH problem, since given a public key $pk = g^K$ and $(c, K) = (g^r, pk^r)$ a *PCO* simply checks whether $(pk = g^K, c = g^r, K = pk^r)$ is a DH-triple, which is exactly a DDH Oracle. It's also perfect (key-)anonymous based on its definition.

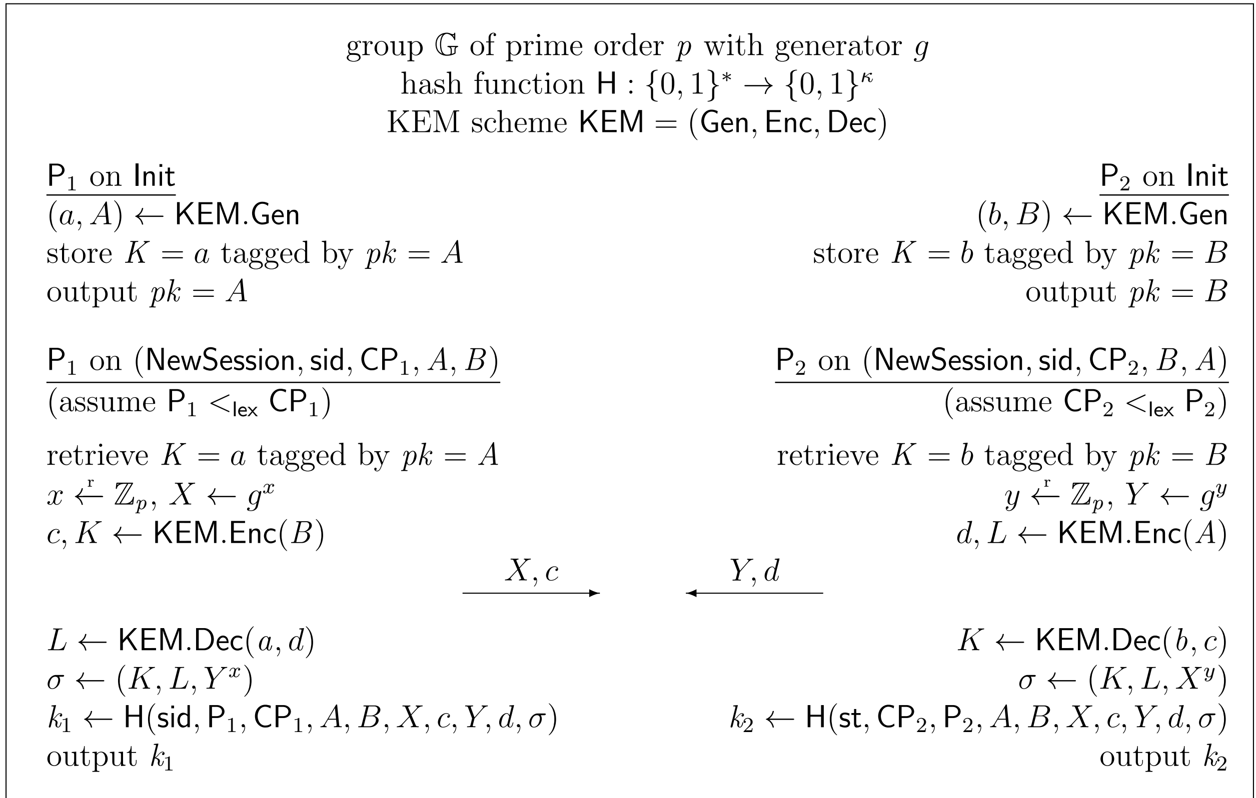


Figure 3.10: Protocol SKEME: KEM-authenticated Key Exchange

Theorem 3.3. *Protocol SKEME shown in Figure 3.10 realizes* $\mathcal{F}_{\mathsf{khAKE}}$ *if the Gap CDH assumption holds,* KEM *is a OW-PCA secure and perfect (key-)anonymous KEM, and* H *is a random oracle.*

Because of inherent similarities of SKEME and 3DH, the proof of the above theorem follows a similar pattern as the proof of Theorem 3.1, which we show below.

Initialization: Initialize empty global list KL and empty lists KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
set $(K, pk) \leftarrow \mathsf{KEM.Gen}$, add (K, pk) to KL and KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (K, pk) \in KL_{\mathsf{P}}$ send K to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \xleftarrow{r} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$
store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w, e, r \rangle$ and send $(W = g^w, e)$ to $\mathcal{A}$

On $\mathcal{A}$'s message (Z, f) to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w, e, \cdot \rangle$:
  if there is *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z, f', \cdot \rangle$ s.t. $g^z = Z$ and $f = f'$ then:
    send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
  send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, (Z, f))$ to $\mathcal{F}$

On query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ to random oracle H, for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:
if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x, c, r \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t.
$(X, \sigma) = (g^x, (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^x))$:
  send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, (Y, d))$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y, d, r \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t.
$(Y, \sigma) = (g^y, (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A, r), X^y))$:
  send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, (X, c))$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 3.11: Simulator SIM showing that SKEME realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

Proof. We use the following definitions for any $\mathsf{P}^{\mathsf{sid}}$, for $\mathsf{P} \in \{\mathsf{C}, \mathsf{S}\}$, which always uses intended counterparty public key, i.e. $\exists\ pk_{\mathsf{CP}}$ s.t. $\mathsf{P}^{\mathsf{sid}}$ runs on $(\mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$:

Suppose that $e, r_{\mathsf{P}}^{\mathsf{sid}} \leftarrow \mathsf{KEM.Enc}_1(\kappa), M \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{CP}}, r_{\mathsf{P}}^{\mathsf{sid}})$ and $f, r_{\mathsf{CP}}^{\mathsf{sid}} \leftarrow \mathsf{KEM.Enc}_1(\kappa)$, $N \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{P}}, r_{\mathsf{CP}}^{\mathsf{sid}})$ are generated by $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ correspondingly. We use $r_{\mathsf{P}}^{\mathsf{sid}}$,$e_{\mathsf{P}}^{\mathsf{sid}}$ and $M_{\mathsf{P}}^{\mathsf{sid}}$ to represent r, e, M locally generated by $\mathsf{P}^{\mathsf{sid}}$ under some $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$.

Let KL be the list of all key pairs generated so far, and KL_{P} be the set of key pairs generated

for P, $KL^+(\mathsf{P}^{\mathsf{sid}})$ stands for $KL \cup \{(K_{\mathsf{CP}}, pk_{\mathsf{CP}})\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$ and K_{CP} is corresponding K which doesn't necessarilly need to be known or verified. (If $(K_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL$ then $KL^+(\mathsf{P}^{\mathsf{sid}}) = KL$). Using these notions, we define following functions for every $\mathsf{P}^{\mathsf{sid}}$:

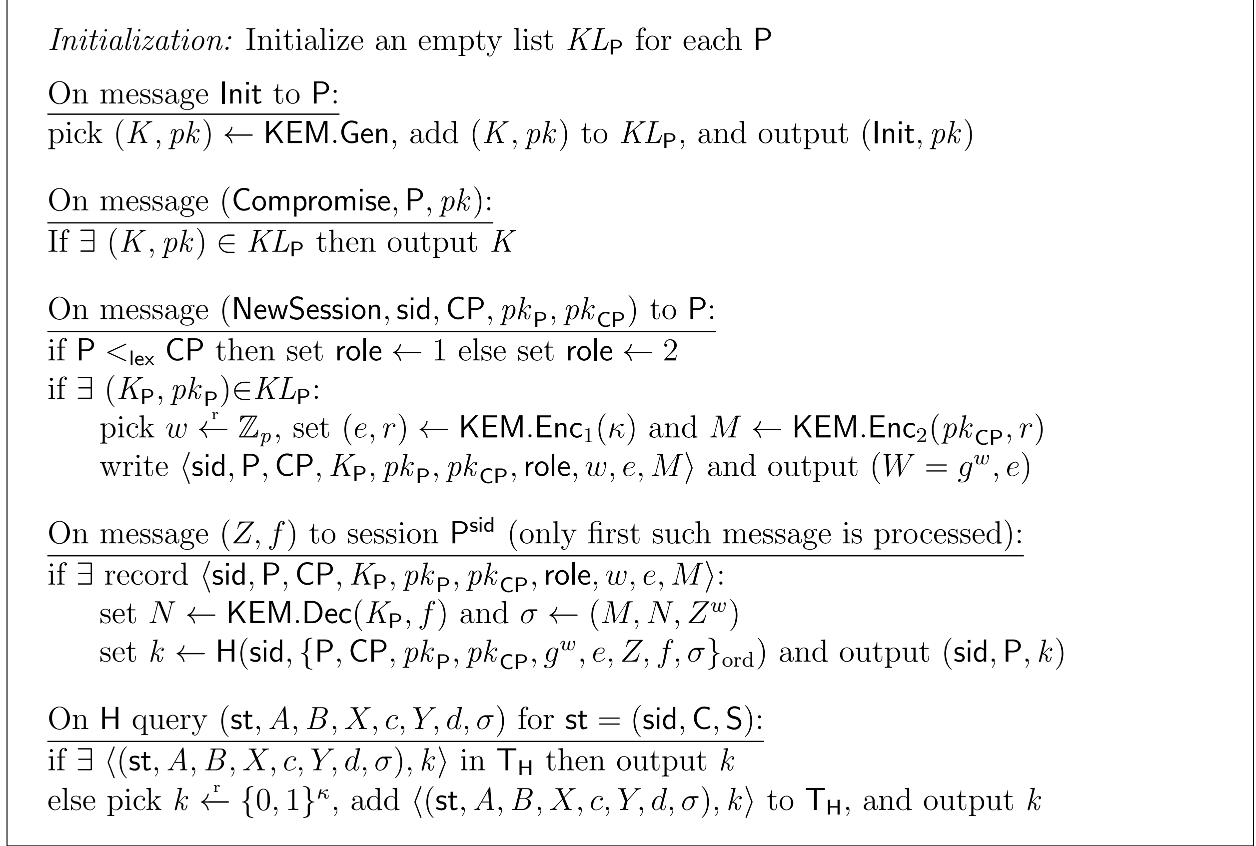
$$\begin{aligned}
\mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', Y, d) = \quad & (\mathsf{KEM.Enc}_2(pk', r), \mathsf{KEM.Dec}(K, d), Y^x) \text{ for} \\
r = r^{\mathsf{sid}}_{\mathsf{C}}, x = x^{\mathsf{sid}}_{\mathsf{C}}, \quad & (K, pk) \in KL_{\mathsf{C}}, (\cdot, pk') \in KL^+(\mathsf{C}^{\mathsf{sid}}) \\
\mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', X, c) = \quad & (\mathsf{KEM.Dec}(K, c), \mathsf{KEM.Enc}_2(pk', r), X^y) \text{ for} \\
r = r^{\mathsf{sid}}_{\mathsf{S}}, y = y^{\mathsf{sid}}_{\mathsf{S}}, \quad & (K, pk) \in KL_{\mathsf{S}}, (\cdot, pk') \in KL^+(\mathsf{S}^{\mathsf{sid}})
\end{aligned}$$

$$\begin{aligned}
R^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', (Y, d)) &= \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, pk', X^{\mathsf{sid}}_{\mathsf{C}}, c^{\mathsf{sid}}_{\mathsf{C}}, Y, d, \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', Y, d)) \\
R^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', (X, c)) &= \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, pk', X, c, Y^{\mathsf{sid}}_{\mathsf{S}}, d^{\mathsf{sid}}_{\mathsf{S}}, \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', X, c))
\end{aligned}$$

GAME 0 *(real world)*: This is the real-world interaction of environment $\mathcal{Z}$ (and its subroutine $\mathcal{A}$) with the SKEME protocol, shown in Fig. 3.12.

GAME 1 *(past* H *queries are irrelevent to new sessions)*: We add an abort if session $\mathsf{P}^{\mathsf{sid}}$ starts with W which appeared in some prior inputs to H. As in the case of 3DH, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$.

GAME 2 *(programming* $R^{\mathsf{sid}}_{\mathsf{P}}$ *values into* H *outputs)*: Define sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ to be *matching* if $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}} = \mathsf{S}$ and $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}} = \mathsf{C}$. By correctness of SKEME for any matching sessions and any public keys A, B it holds that $R^{\mathsf{sid}}_{\mathsf{C}}(A, B, (Y_{\mathsf{S}}, d_{\mathsf{S}})) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X_{\mathsf{C}}, c_{\mathsf{C}}))$. In Game 2 we set H outputs using functions $R^{\mathsf{sid}}_{\mathsf{P}}$. For every pair of matching sessions $(\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}})$ we consider a



Initialization: Initialize an empty list KL_{P} for each P

On message Init to P:
pick $(K, pk) \leftarrow \mathsf{KEM.Gen}$, add (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
if $\exists\ (K_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$:
  pick $w \overset{r}{\leftarrow} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$ and $M \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{CP}}, r)$
  write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, M \rangle$ and output $(W = g^w, e)$

On message (Z, f) to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, M \rangle$:
  set $N \leftarrow \mathsf{KEM.Dec}(K_{\mathsf{P}}, f)$ and $\sigma \leftarrow (M, N, Z^w)$
  set $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, g^w, e, Z, f, \sigma\}_{\text{ord}})$ and output $(\mathsf{sid}, \mathsf{P}, k)$

On H query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:
if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k
else pick $k \overset{r}{\leftarrow} \{0,1\}^\kappa$, add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$, and output k

Figure 3.12: SKEME: Environment's view of real-world interaction (Game 0)

pair of random functions $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$ s.t. for all A, B

$$R_{\mathsf{C}}^{\mathsf{sid}}(A, B, (Y_{\mathsf{S}}^{\mathsf{sid}}, d_{\mathsf{S}}^{\mathsf{sid}})) = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, (X_{\mathsf{C}}^{\mathsf{sid}}, c_{\mathsf{C}}^{\mathsf{sid}})) \tag{3.5}$$

Since they're matching sessions, above equation satisfy $c_{\mathsf{S}}^{\mathsf{sid}} = c_{\mathsf{C}}^{\mathsf{sid}}, d_{\mathsf{C}}^{\mathsf{sid}} = d_{\mathsf{S}}^{\mathsf{sid}}$.

More precisely, for any session $\mathsf{P}^{\mathsf{sid}}$ with no matching session $R_{\mathsf{P}}^{\mathsf{sid}}$ is set as a random function, and for $\mathsf{P}^{\mathsf{sid}}$ for which a matching session exists $R_{\mathsf{P}}^{\mathsf{sid}}$ is set as a random function subject to constraint (3.5). Consider an oracle H which responds to each new query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ as follows:

1. If $\exists \mathsf{C}^{\mathsf{sid}}$ s.t. $(\mathsf{S}, X) = (\mathsf{CP}_{\mathsf{C}}^{\mathsf{sid}}, X_{\mathsf{C}}^{\mathsf{sid}})$, and $\exists\ A, B$ s.t. $(\cdot, A) \in KL_{\mathsf{C}}, (\cdot, B) \in KL^+(\mathsf{C}^{\mathsf{sid}})$ and $\sigma = \mathsf{SKEME}_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y, d)$: Set $k \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, (Y, d))$

2. If $\exists \mathsf{S}^{\mathsf{sid}}$ s.t. $(\mathsf{C}, Y) = (\mathsf{CP}_{\mathsf{S}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\ B, A$ s.t. $(\cdot, B) \in KL_{\mathsf{S}}$, $(\cdot, A) \in KL^+(\mathsf{S}^{\mathsf{sid}})$ and

Initialization: Initialize empty lists: PK, CPK, KL and KL_{P} for all P

<u>On message Init to P:</u>

set $(K, pk) \leftarrow \mathsf{KEM.Gen}$, send (Init, pk) to P, add pk to PK and (K, pk) to KL and KL_{P}

<u>On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:</u>

If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ add pk to CPK and output K

<u>On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:</u>

if $\exists\ (K, pk_{\mathsf{P}}) \in KL_P$ then:
- initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$
- if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
- pick $w \xleftarrow{r} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$
- write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, \perp \rangle$ as fresh, output $(W = g^w, e)$

<u>On message (Z, f) to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):</u>

if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, \perp \rangle$:
- if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \cdot, z, f', \cdot, k' \rangle$ s.t. $g^z = Z$ and $f = f'$
  - then if rec' is fresh, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$ and $k' \neq \perp$:
    - then $k \leftarrow k'$
    - else $k \xleftarrow{r} \{0,1\}^\kappa$
  - else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ and re-label rec as $\mathsf{interfered}$
- update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$

<u>On H query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:</u>

if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, c, r, \cdot \rangle$ s.t.
   $(X, \sigma) = (g^x, (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^x))$ for some $(a, A) \in KL_{\mathsf{C}}$ and B s.t. $B \in CPK$ or $B \notin PK$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, (Y, d))$

2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, d, r, \cdot \rangle$ s.t.
   $(Y, \sigma) = (g^y, (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A, r), X^y))$ for some $(b, B) \in KL_{\mathsf{S}}$ and A s.t. $A \in CPK$ or $A \notin PK$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X, c))$

add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 3.13: SKEME: Environment's view of ideal-world interaction (Game 7)

$\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{S}}(B, A, X, c)$: Set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X, c))$

3. In any other case pick $k \xleftarrow{r} \{0,1\}^\kappa$

Since the game knows each key pair $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ generated for each P, randomness r used in $\mathsf{KEM.Enc}$, and the ephemeral state w of each session $\mathsf{P}^{\mathsf{sid}}$, it can decide for any Z, f, pk' whether $\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk', Z, f)$ if pk' is honestly generated, i.e. $(K', pk') \in KL$. Moreover, even if pk' is adversarially generated, i.e. $(K', pk') \in KL^+(\mathsf{P}^{\mathsf{sid}}) \setminus KL$, the game can still decide, since it records r and can generate M via $\mathsf{KEM.Enc}_2(pk', r)$, it can instead

check whether the corersponding part of σ equals to M. Note that each value of $R_{\mathsf{P}}^{\mathsf{sid}}$ is used to program H on at most one query. Also, if H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ satisfies both conditions then $(X, Y) = (g^x, g^y)$ and $(c, d) = (c_{\mathsf{C}}^{\mathsf{sid}}, d_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\ A', B', a, b$ s.t.

$$\mathsf{SKEME}_{\mathsf{C}}^{\mathsf{sid}}(g^a, B', Y, d) = (\mathsf{KEM.Enc}_2(B', r_{\mathsf{C}}^{\mathsf{sid}}), \mathsf{KEM.Dec}(a, d), Y^x)$$

$$= (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A', r_{\mathsf{S}}^{\mathsf{sid}}), X^y) = \mathsf{SKEME}_{\mathsf{S}}^{\mathsf{sid}}(g^b, A', X, c)$$

Since by security of $\mathsf{KEM.Enc}_2$ the above equations imply that $(A', B') = (A, B)$(e.g.$\mathsf{KEM.Dec}(b, c) = \mathsf{KEM.Enc}_2(B, r_{\mathsf{C}}^{\mathsf{sid}}) = \mathsf{KEM.Enc}_2(B', r_{\mathsf{C}}^{\mathsf{sid}})$), and by (3.5) we already know that $R_{\mathsf{C}}^{\mathsf{sid}}(A, B, (Y_{\mathsf{S}}^{\mathsf{sid}}, d_{\mathsf{S}}^{\mathsf{sid}})) = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, (X_{\mathsf{C}}^{\mathsf{sid}}, c_{\mathsf{C}}^{\mathsf{sid}}))$, it follows that if both conditions are satisfied then both program H output to the same value. Thus we conclude:

$$\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]$$

GAME 3 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: As in 3DH, in Game 3 we make the following changes: We mark each initialized session $\mathsf{P}^{\mathsf{sid}}$ as fresh, and when $\mathcal{A}$ sends (Z, f) to $\mathsf{P}^{\mathsf{sid}}$ then it re-labels $\mathsf{P}^{\mathsf{sid}}$ as interfered unless (Z, f) equals to the message sent by the intended counterparty of P. In other words, $\mathsf{C}^{\mathsf{sid}}$ is re-labeled interfered if $Z_{\mathsf{C}}^{\mathsf{sid}} \neq Y_{\mathsf{S}}^{\mathsf{sid}}$ or $f_{\mathsf{C}}^{\mathsf{sid}} \neq d_{\mathsf{S}}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ is re-labeled interfered if $Z_{\mathsf{S}}^{\mathsf{sid}} \neq X_{\mathsf{C}}^{\mathsf{sid}}$ or $f_{\mathsf{S}}^{\mathsf{sid}} \neq c_{\mathsf{C}}^{\mathsf{sid}}$. Secondly, we say session $\mathsf{P}^{\mathsf{sid}}$ runs "under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$" if it runs on its own key pair $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ and intended counterparty public key pk_{CP}. Using this notation Game 3 computes $k_{\mathsf{P}}^{\mathsf{sid}}$ as follows:

1. If $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ are *matching*, both are fresh, $\mathsf{CP}^{\mathsf{sid}}$ runs under $(pk_{\mathsf{CP}}, pk_{\mathsf{P}})$, and $k_{\mathsf{CP}}^{\mathsf{sid}} \neq \perp$, then $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow k_{\mathsf{CP}}^{\mathsf{sid}}$

2. In all other cases set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$

We argue that this change makes no difference to the environment. In Game 2 value $k_{\mathsf{P}}^{\mathsf{sid}}$ is derived from $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, W, e_{\mathsf{P}}^{\mathsf{sid}}, Z, f\}_{\text{ord}}, \sigma)$, where $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$. However, H is programmed in Game 2 to output $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ if $\exists$ (Z, f) s.t. $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ for any $(\cdot, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. Since pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ must be in set $KL^{+}(\mathsf{P}^{\mathsf{sid}})$, setting $k_{\mathsf{P}}^{\mathsf{sid}}$ directly as $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ only short-circuits this process. Moreover, since $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ are correlated by equation (3.5), setting $k_{\mathsf{C}}^{\mathsf{sid}}$ as $k_{\mathsf{S}}^{\mathsf{sid}}$ or vice versa, in the case both are fresh, does not change the game. Thus we conclude:

$$\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$$

GAME 4 *(abort on* H *queries for passive sessions)*: We add an abort if oracle H triggers evaluation of $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', (Z, f))$ for any pk, pk' and $(Z, f) = (W_{\mathsf{CP}}^{\mathsf{sid}}, e_{\mathsf{CP}}^{\mathsf{sid}})$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$. Note that if $\mathsf{P}^{\mathsf{sid}}$ is passively observed, then value $(W_{\mathsf{CP}}^{\mathsf{sid}}, e_{\mathsf{CP}}^{\mathsf{sid}})$ either has been delivered to $\mathsf{P}^{\mathsf{sid}}$, i.e. $(Z_{\mathsf{P}}^{\mathsf{sid}}, f_{\mathsf{P}}^{\mathsf{sid}}) = (W_{\mathsf{CP}}^{\mathsf{sid}}, e_{\mathsf{CP}}^{\mathsf{sid}})$, or $\mathsf{P}^{\mathsf{sid}}$ is still waiting for message Z. By the code of oracle H in Game 2 the call to $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', (W_{\mathsf{CP}}^{\mathsf{sid}}, e_{\mathsf{CP}}^{\mathsf{sid}}))$ is triggered only if H query $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, W, e, Z, f, \sigma\}_{\text{ord}})$ satisfies the following for $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ and $f = e_{\mathsf{CP}}^{\mathsf{sid}}$:

$$\sigma \;=\; \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z, f)$$

As in proof of 3DH, the hardness of computing σ relies on hardness of computing $\mathsf{cdh}_g(W, Z)$, which is the last element in σ. We show that solving Gap CDH can be reduced to causing event Bad, defined as event that such query happens. The reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in a message of all simulated parties in a randomized way: on $\mathsf{NewSession}$ to P, if $\mathsf{role} = 1$ then $\mathcal{R}$ sends $X = \bar{X}^s$ as the message from $\mathsf{C}^{\mathsf{sid}}$ for $s \leftarrow \mathbb{Z}_p$, and if $\mathsf{role} = 2$ then $\mathcal{R}$ sends $Y = \bar{Y}^t$ as the message from $\mathsf{S}^{\mathsf{sid}}$ for $t \leftarrow \mathbb{Z}_p$. Otherwise it emulates the security game, in particular it knows all of the key pairs (a, A) and (b, B). Let KL be

the list of all key pairs generated so far, and KL_{P} be the set of key pairs generated for P. Although $\mathcal{R}$ doesn't know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, $\mathcal{R}$ can use DDH oracle to emulate the way Game 3 services H queries: To test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ for $X = \bar{X}^s$ satisfies $\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y, d)$, because $\mathcal{R}$ knows all $(K, pk) \in KL$ and records locally generated r, it can check first two part of $\sigma = (K, L, V)$.Then to test if $V = Y^x$, reduction $\mathcal{R}$ checks if $V = \mathsf{cdh}_g(\bar{X}, Y^s)$. Symmetrically on server side, to test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ for $Y = \bar{Y}^t$ satisfies $V = X^y$, reduction $\mathcal{R}$ checks if $V = \mathsf{cdh}_g(X^t, \bar{Y})$. Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3, and if it does $\mathcal{R}$ detects it because it occurs if both above conditions hold, in which case $\mathcal{R}$ outputs $V^{1/(st)}$ as $\mathsf{cdh}_g(\bar{X}, \bar{Y})$. It follows that $\Pr[\mathsf{Bad}] \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$, thus we conclude:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Game 5 *(random keys on passively observed sessions)*: Same as in 3DH, if session $\mathsf{S}^{\mathsf{sid}}$ remains fresh when $\mathcal{A}$ sends (Z, f) to $\mathsf{P}^{\mathsf{sid}}$ then instead of setting $k^{\mathsf{sid}}_{\mathsf{P}} \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ as in Game 3, we now set $k^{\mathsf{sid}}_{\mathsf{P}} \leftarrow \{0,1\}^{\kappa}$. Since by Game 4 oracle H never queries $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ on (Z, f) sent from $\mathsf{P}^{\mathsf{sid}}$'s counterparty, which is a condition for $\mathsf{P}^{\mathsf{sid}}$ to remain fresh, it follows by randomness of $R^{\mathsf{sid}}_{\mathsf{P}}$ that the modified game remains externally identical, hence:

$$\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$$

Game 6 *(decorrelating function pairs $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}}$)*: This game is same as in 3DH, and

$$\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$$

GAME 7 *(hash computation consistent only for compromised keys)*: Recall that in Game 6, as in Game 2, $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \cdot, \cdot, W_{\mathsf{P}}^{\mathsf{sid}}, Z, \sigma\}_{\text{ord}})$ is defined as $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ if $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ for some $(K_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$,$(K_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. In Game 7 we add a condition that this programming of H can occur only if (1)$(K_{\mathsf{CP}}, pk_{\mathsf{CP}})$ is honestly generated and $\mathsf{compromised}$ or (2)$(K_{\mathsf{CP}}, pk_{\mathsf{CP}})$ is adversarially generated and pk_{CP} is the counterparty key $\mathsf{P}^{\mathsf{sid}}$ runs under. In both cases adversary can know K_{CP}.

Let $CPKL$ be the list of generated key pairs that were compromised so far, Game 7 diverges from Game 6 if bad event occurs where H is queried on inputs as above for $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ and $(K_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL \setminus CPKL$, i.e. honestly generated but compromised key pairs, while in Game 6, as in Game 2, this programming was done whenever $(K_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. As in the case of 3DH proof we only argue for client side since the other case is symmetric. We define event Bad, where in client side's H query value $\sigma = (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^x)$ for some $(a, A) \in KL_{\mathsf{C}}$ and $(b, B) \in KL \setminus CPKL$ which is fresh.

We show a reduction $\mathcal{R}$ that breaks $OWPCA$ security if Bad occurs. On input a $OWPCA$ challenge $(\bar{B}, c^*)$, $\mathcal{R}$ has access to $PCO_K(.,.)$ whose inner K corresponds to $\bar{B}$, and $\mathcal{R}$ doesn't know randomness r used to generate c^*. $\mathcal{R}$ sets each $X_{\mathsf{C}}^{\mathsf{sid}}$ as g^x for random x and each $Y_{\mathsf{S}}^{\mathsf{sid}}$ as g^y for random y. $\mathcal{R}$ also picks all key pairs except that in the i-th session, for a chosen index $j \in [1, \ldots, q_{\text{K}}]$, where $\mathcal{R}$ set the j-th public key pk_{CP} as $\bar{B}$, and sets c as c^*. Let $\mathsf{Bad}_{i,j}$ denote Bad occurring for this j-th public key in the i-th session, i.e. $pk_{\mathsf{CP}} = \bar{B}$.

As long as the corresponding K_{CP} is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to compromise of all other keys, and serve H queries as follows: To test server side H query input $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma)$, i.e. if $\sigma = (K, L, V)$, $\mathcal{R}$ tests as in Game 6 except for b that corrsponds to the public key $\bar{B}$, in which case $\mathcal{R}$ tests if $K = \mathsf{KEM.Enc}_2(pk, r) = \mathsf{KEM.Dec}(K, c)$ via checking if $PCO_K(K, c)$ returns 1. To test client side, i.e. if $\sigma = (K, L, V)$ for any pk including $pk = \bar{B}$, $\mathcal{R}$ also tests as in Game 6, except for the case that K is the private key corresponding to the public key $\bar{B}$, in which

case $\mathcal{R}$ replaces testing $K = \mathsf{KEM.Enc}_2(pk, r)$ with checking if $PCO_K(K, c)$ returns 1.

$\mathsf{Bad}_{i,j}$ can happen only before $(K_{\mathsf{CP}}, pk_{\mathsf{CP}})$ used in that session is compromised, so it occurs in reduction with same probability as in Game 6. $\mathcal{R}$ can detect event $\mathsf{Bad}_{i,j}$ because it occurs if H query involves the j-th credential and on client side, given c and $\bar{B}$, it can output correct K that satisfies $K = \mathsf{KEM.Enc}_2(\bar{B}, r) = \mathsf{KEM.Dec}(b, c)$, without knowing the value of r and b, in which case it outputs correct K corresponding to c^* and breaks $OWPCA$ security. If $\mathcal{R}$ picks index i and j at random it follows that $\Pr[\mathsf{Bad}] \leq q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$.
Since a symmetric argument holds also for server side, we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_{\mathrm{K}}) \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$$

Observe that Game 7 is identical to the ideal-world game shown in Figure 3.12: By Game 6 all functions $R^{\mathsf{sid}}_{\mathsf{P}}$ are random, by Game 5 the game responds to (Z, f) messages to $\mathsf{P}^{\mathsf{sid}}$ as the game in Figure 3.12, and after the modification in oracle H done in Game 7 this oracle also acts as in Figure 3.12. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 3.3.

□

3.6 Compiler from key-hiding AKE to aPAKE

We show that any UC Key-Hiding AKE protocol can be converted to a UC asymmetric PAKE (aPAKE) with a very small computational overhead. We call this AKE-to-aPAKE compiler construction KHAPE, which stands for Key-Hiding Asymmetric PakE, shown in Figure 3.14. The compiler views each party's AKE inputs, namely its own private key

and its counterparty public key, as a single object, an AKE "credential". The two parties participating in aPAKE, the server and the user, a.k.a. the client, each will have such a credential: The server's credential contains the server's private key and the client's public key, and the client's credential contains the client's private key and the server's public key. Running AKE on such matching pair of inputs would establish a secure shared key, but while the server can store its credential, the client's only input is her password and it is not clear how one can derive an AKE credential from a password. Protocol KHAPE enables precisely this derivation: In addition to server's credential, the server will also store a ciphertext which encrypts, via an ideal cipher, the client's credential under the user's password, and the aPAKE protocol consists of server sending that ciphertext to the client, the client decrypting it using the user's password to obtain its certificate, and using that certificate to run an AKE instance with the server.

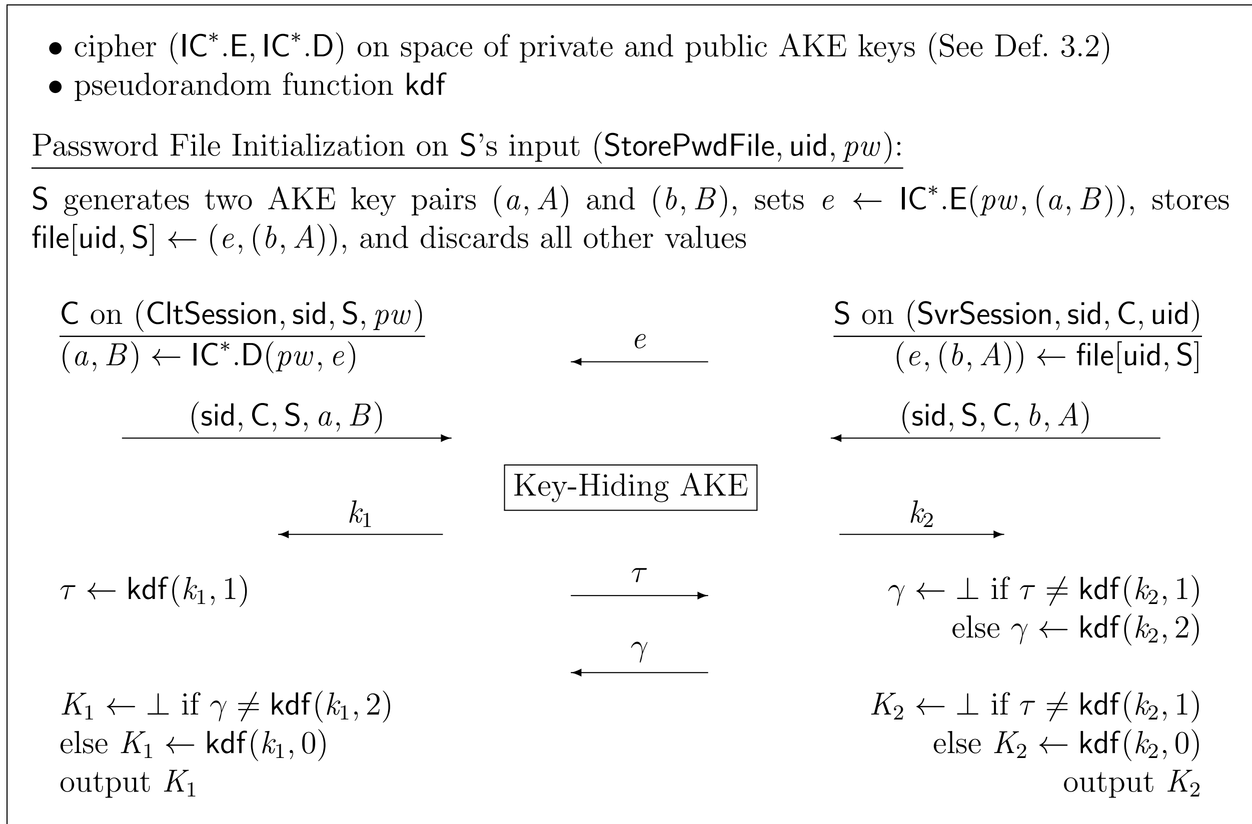


Figure 3.14: Protocol KHAPE: Compiler from Key-Hiding AKE to aPAKE

Reduced-bandwidth variant. In the aPAKE construction in Figure 3.14, ciphertext e password-encrypts a pair of the client's secret key K_{C} and the server's public key pk_{S}. Without loss of generality every AKE key pair (K, pk) is generated by the key generation algorithm from uniformly sampled randomness r. The aPAKE construction can be modified so that envelope e password-encrypts only the server's public key pk_{S}, while the client derives its private key K_{C} using the key generation algorithm on randomness $r \leftarrow \mathsf{H}(pw)$ via RO hash H. Note that if key-hiding AKE is either 3DH or HMQV then this amounts to the client setting it's secret exponent $a \leftarrow \mathsf{H}(pw)$ where H maps onto range $\mathbb{Z}_q$.[7] This change does not simplify the construction of the ideal cipher by much because typically the public key is a group element and the private key is a random modular residue, but it reduces the size of ciphertext e. We believe that the security proof for the aPAKE protocol in Figure 3.14 can be adjusted to show security of this reduced-bandwidth implementation.

Why we need key-hiding AKE. Note that anyone who observes the credential-encrypting ciphertext e can decrypt it under any password. Each password guess will decrypt e into some credential $cred = (K_{\mathsf{C}}, pk_{\mathsf{S}})$, where K_{C} is a client's private key and pk_{S} is a server's public key. Let $cred(pw)$ denote the credential obtained by decrypting e using password pw. For any password guess pw^* the attacker can use credential $cred(pw^*)$ as input to an AKE protocol with the server, but that is equivalent to an on-line password authentication attempt using pw^* as a password guess (see below). Note that the attacker can also either watch or interfere with AKE instances executed by the honest user on credential $cred(pw)$ that corresponds to the correct password pw. Moreover, the attacker w.l.o.g. holds a list of credential candidates $cred(pw_1), ..., cred(pw_n)$ corresponding to offline password guesses. However, the key-hiding property of AKE implies that even if $cred(pw)$ is on the attacker's list, interfering or watching client's AKE instances cannot help the attacker decide which credential is the one that the client uses. The only way to learn anything from client AKE

[7] If AKE is implemented as SKEME of Section 3.5 then the client must also derive the public key pk_{C}, since it is used in the key-derivation hash, see Figure 3.10.

instances on input $cred(pw)$ would be to engage them using a matching credential, i.e. $(K_{\mathsf{S}}, pk_{\mathsf{C}})$. This is possible if the adversary compromises the server who holds exactly these keys, but otherwise doing so is equivalent to breaking AKE security.

Why we need mutual key confirmation. To handle the server-side attack we needed the key-hiding property of AKE to imply that the only way to decide which keys $(K_{\mathsf{S}}, pk_{\mathsf{C}})$ the server uses is to engage in an AKE instance using the matching counterparty keys (K_{C}, pk_S). The key-hiding property provided by 3DH and HMQV, as modeled by functionality $\mathcal{F}_{\mathsf{khAKE}}$, actually does *not* suffice for this by itself. Let the attacker hold a list of n possible decrypted client credentials $cred_i = cred(pw_i) = (a_i, B_i)$ for $i = 1, ..., n$, and let S hold credential $cred_{\mathsf{S}} = (b, A)$ which matches $cred_i$, i.e. $A = g^{a_i}$ and $B_i = g^b$, which is the case if password guess pw_i matches the correct password pw. If an active attacker chooses x and sends $X = g^x$ to S then it can locally complete the 3DH or HMQV equation using *any* key pair (a_i, B_i) it holds, thus computing n candidate session keys k_i. By 3DH or HMQV correctness, since the i-th client credential matches the server's credential, key k_i equals to the session key k computed by S. Therefore, if S used key k straight away then the attacker could observe that $k_i = k$ and hence that $pw_i = pw$.

However, the fix is simple: To make the server's session key output safe to use, the client must first send a key confirmation message to the server, implemented in Figure 3.14 by client's final message τ. This stops the attack because the attacker sending τ uniquely determines one of the keys k_i on its candidate list, and since this succeeds only if $k_i = k$, this attack reduces to an on-line test of a single password guess pw_i, which is unavoidable in a (a)PAKE protocol. A natural question is if there is no equivalent attack on the client-side, which would be abetted by the client sending a key confirmation message τ. This is not the case because of the following asymmetry: Off-line password guesses give the attacker a list of possible *client-side* credentials, which by AKE rules can be tested against server sessions. However, by the the key-hiding property of AKE such credentials are useless in deciding

which of them, if any, is used by the honest user. Moreover, since the ciphertext e encrypts only the client-side keys, by the KCI property of the AKE the knowledge of client-side keys is not helpful in breaking the security of AKE instances executed by the honest client on such keys.

Server-to-client key confirmation is needed too, in this case to ensure forward secrecy. Without it, an attacker could choose $Y = g^y$ (in the HMQV or 3DH instantiations) and later, after the session is complete, compromise the server to learn the private key b with which it can compute the session key. The client-to-server key confirmation addresses this issue on the client side.

In addition to ensuring security, key confirmation serves as (explicit) *entity authentication* in this aPAKE construction.

Why we need credential encryption to be an ideal cipher. Note that the attacker can attack the client too, by sending an arbitrary ciphertext to the client, but the ideal cipher property is that the ciphertext commits the attacker to only one choice of key for which the attacker can decide a plaintext: for all other keys the decrypted plaintext will be random.

For the above to work the encryption used to password-encrypt the client credential needs to be an ideal cipher over the space of (private,public) key pairs used in AKE. In all key-hiding AKE protocols examples we discuss in this paper, i.e. 3DH, HMQV, as well as SKEME instantiated with Diffie-Hellman KEM, this message space is $\mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p. We refer to Section 3.8 for several methods of instantiate an ideal cipher on this space. Here we will assume the implementation of the following form, which is realized by the Elligator2 or Elligator-squared encodings (see Section 3.8).

Definition 3.2. [($\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D}$) instantiation.] *Let X be the Cartesian product of the space of private keys and the space of public keys in AKE, let $\mathsf{IC.E}, \mathsf{IC.D}$ be an ideal cipher on n-bit strings, and let map be a (randomized) invertible quasi-bijective map from X to $X' = \{0,1\}^n$.*

A randomized 1-1 function $\mathsf{map} : X \rightarrow X'$ *is quasi-bijective if there is a negligible statistical difference between a uniform distribution over* X' *and* $x' \xleftarrow{r} \mathsf{map}(x)$ *for random* x *in* X*. Instead of a direct ideal cipher on message space* X *protocol* KHAPE *in Fig. 3.14 uses a randomized cipher* $(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ *on* X' *where* $\mathsf{IC}^*.\mathsf{E}(x)$ *outputs* $\mathsf{IC}.\mathsf{E}(x')$ *where* $x' \leftarrow \mathsf{map}(x; r)$ *for random* r *used by* map*, and* $\mathsf{IC}^*.\mathsf{D}(y)$ *outputs* $x = \mathsf{map}^{-1}(x')$ *where* $x' = \mathsf{IC}.\mathsf{D}(y)$*.*

Comparison with Encrypted Key Exchange of Bellovin-Merritt. It is instructive to compare the KHAPE design to that of the "Encrypted Key Exchange" (EKE) construction of Bellovin-Meritt [28]. The EKE compiler starts from unauthenticated KE, uses an Ideal Cipher to encrypt each KE protocol message under the password, and this results in UC PAKE in the IC model (see e.g. [109]). By contrast, our compiler starts from *Authenticated* KE, and uses IC to password-encrypt only the client's inputs to the AKE protocol, while the protocol messages themselves are exchanged without any change. Just like EKE, our compiler adds only symmetric-key overhead to the underlying KE, but it results in an aPAKE instead of just PAKE. However, just like EKE, it imposes additional requirements on the underlying key exchange protocol: Whereas EKE needs the key exchange to have a "random transcript" property, i.e. KE protocol messages must be random in some message space, in the case of KHAPE the underlying AKE needs to have the key-hiding property we define in Section 4.2. Either condition also relies on an Ideal Cipher (IC) modeling for a non-standard plaintext space: For EKE the IC plaintext space is the space of KE protocol *messages*, while for KHAPE the IC plaintext space is the Cartesian product of the space of private keys and the space of public keys which form AKE protocol *inputs*.

UC aPAKE security model. We refer to Section 2.3.3 for the functionality $\mathcal{F}_{\mathsf{aPAKE}}$ we use to model UC aPAKE. This model is very similar to the one defined by Gentry et al. [72] with some modifications that we discuss in Section 2.3.3. The main notational change is that we use a *user account identifier* uid, instead of generic *session identifier* sid, to index password files held by a given server. Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ also includes uni-directional

(client-to-server) entity authentication as part of the security definition. We refer to Section 2.3.3 also for a discussion of several subtle issues involved in UC modeling of tight bounds on adversary's local computation during an offline dictionary attack.

Theorem 3.4. *Protocol* KHAPE *realizes the UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *if the AKE protocol realizes the Key-Hiding AKE functionality* $\mathcal{F}_{\mathsf{khAKE}}$*, assuming that* kdf *is a secure PRF and* $(\mathsf{Enc}, \mathsf{Dec})$ *is an ideal cipher over message space of private,public key pairs in AKE.*

Initialization

Initialize simulator $\mathsf{SIM}_{\mathsf{AKE}}$, an empty table T_{IC}, empty lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$

Notation: $\mathsf{T}_{\mathsf{IC}}^{pw}.X' = \{x' \| \ \exists y \ (pw, x', y) \in \mathsf{T}_{\mathsf{IC}}\}$, $\mathsf{T}_{\mathsf{IC}}^{pw}.Y = \{y \mid \exists x' \ (pw, x', y) \in \mathsf{T}_{\mathsf{IC}}\}$

Convention: First call to SvrSession or StealPwdFile for $(\mathsf{S}, \mathsf{uid})$ sets $e_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} Y$

Ideal Cipher IC queries

- On query (pw, x') to IC.E, send back y if $(pw, x', y) \in \mathsf{T}_{\mathsf{IC}}$, otherwise pick $y \xleftarrow{r} Y \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.Y$, add (pw, x', y) to T_{IC}, and send back y
- On query (pw, y) to IC.D, send back x' if $(pw, x', y) \in \mathsf{T}_{\mathsf{IC}}$, otherwise do:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \xleftarrow{r} X' \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and:
     (a) If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" then set $(A, B) \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     (b) Otherwise initialize keys A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}

     Set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, B)$, send query $(\mathsf{Compromise}, A)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define a as $\mathsf{SIM}_{\mathsf{AKE}}$'s response, add A to CPK, set $x' \xleftarrow{r} \mathsf{map}(a, B)$

  In either case add (pw, x', y) to T_{IC} and send back x'

Stealing Password Data

On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$. If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "no password file," pass it to $\mathcal{A}$, otherwise declare $(\mathsf{S}, \mathsf{uid})$ compromised and:

1. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns no value then initialize keys A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}
2. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns pw then set $(A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw)$

Send $(\mathsf{Compromise}, B)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define b as $\mathsf{SIM}_{\mathsf{AKE}}$'s response, add B to CPK, set $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$ to $\mathcal{A}$.

Figure 3.15: Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 1

Starting AKE sessions

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$, initialize random function $R_{\mathsf{S}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $e_{\mathsf{S}}^{\mathsf{uid}}$ to $\mathcal{A}$ as message from $\mathsf{S}^{\mathsf{sid}}$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$.

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$ and message e' sent by $\mathcal{A}$ to $\mathsf{C}^{\mathsf{sid}}$, initialize random function $R_{\mathsf{C}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$, and:

1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ check if e' was output by $\mathsf{IC.E}$ on some (pw, x'), and:
   (a) If there is no such $\mathsf{IC.E}$ query then send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
   (b) Otherwise define (pw, x') as the first query to $\mathsf{IC.E}$ which outputted e', send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, and:
      i. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "wrong guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
      ii. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess": set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$ and run the AKE protocol on behalf of $\mathsf{C}^{\mathsf{sid}}$ on inputs $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$; When $\mathsf{C}^{\mathsf{sid}}$ terminates with key k then send $\tau \leftarrow \mathsf{kdf}(k, 1)$ to $\mathcal{A}$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$

Responding to AKE messages

SIM forwards AKE protocol messages between $\mathcal{A}$ and $\mathsf{SIM}_{\mathsf{AKE}}$, and reacts as follows to $\mathsf{SIM}_{\mathsf{AKE}}$'s queries to $\mathcal{F}_{\mathsf{khAKE}}$, whose role is played by SIM. (SIM ignores $\mathsf{SIM}_{\mathsf{AKE}}$'s queries pertaining to any $\mathsf{P}^{\mathsf{sid}}$ that was not started by a $\mathsf{NewSession}$ message.)

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$ set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$ and $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ then send $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$;

   If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" output $\tau \leftarrow \mathsf{kdf}(k, 1)$ for $k = R_{\mathsf{C}}^{\mathsf{sid}}(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}, \alpha)$
2. In any other case (including "wrong guess" above), output $\tau \xleftarrow{\mathrm{r}} \{0,1\}^\kappa$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and $\mathcal{A}$ sends τ' to $\mathsf{S}^{\mathsf{sid}}$:

1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by SIM for $\mathsf{C}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and output $\gamma \xleftarrow{\mathrm{r}} \{0,1\}^\kappa$
2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\tau' = \mathsf{kdf}(k,1)$ for $k = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, \alpha)$ and $(A, B) = pk_{\mathsf{S}}^{\mathsf{uid}}(pw)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, pw), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$, output $\gamma \leftarrow \mathsf{kdf}(k, 2)$
3. Else $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, \perp), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, output $\gamma \xleftarrow{\mathrm{r}} \{0,1\}^\kappa$

If $\mathsf{SIM}_{\mathsf{AKE}}$ sends γ' to $\mathsf{C}^{\mathsf{sid}}$:

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ and γ' was generated by SIM for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$
2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$, $\mathcal{F}_{\mathsf{aPAKE}}$ sent "correct guess" for $\mathsf{C}^{\mathsf{sid}}$, and $\gamma' = \mathsf{kdf}(k, 2)$ for k computed for $\mathsf{C}^{\mathsf{sid}}$ above, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$
3. Else send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$:

If $pk \in PK_{\mathsf{P}}$ and $pk' \in CPK$ send $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ to $\mathcal{A}$

Figure 3.16: Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 2

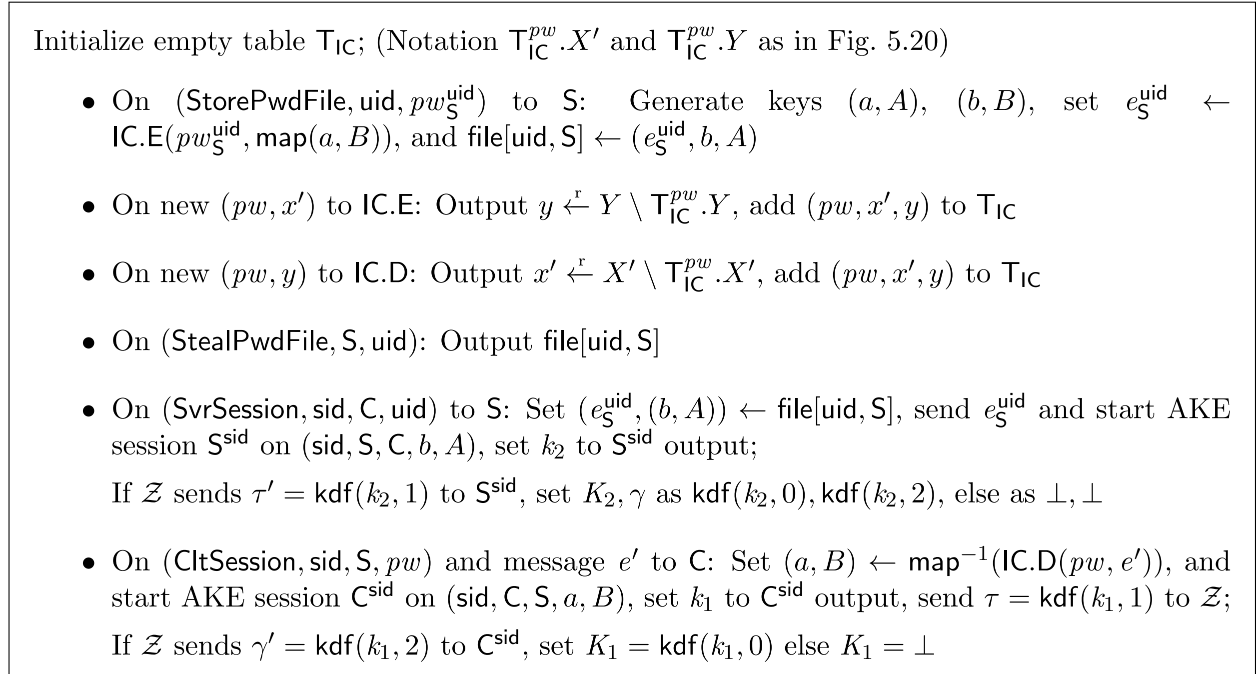
We show that the environment's view of the *real-world* security game, denoted Game 0, i.e. an interaction between the real-world adversary and honest parties who follow protocol KHAPE, is indistinguishable from the environment's view of the *ideal-world* game, denoted Game 8, i.e. an interaction between simulator SIM of Figures 5.20 and 4.19 and functionality $\mathcal{F}_{\mathsf{aPAKE}}$. As before, we use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G8}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let q_{pw}, q_{IC}, and q_{ses} be the upper-bounds on the number of resp. password files, IC queries, and online S or C aPAKE sessions. Let $\epsilon^{\mathcal{Z}}_{\text{kdf}}(\mathsf{SIM}_{\mathsf{AKE}})$ and $\epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$ be the advantages of an environment who uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{IC}} + q_{\text{ses}} + q_{\mathsf{pw}})$ exponentiations in $\mathbb{G}$ in resp. breaking the PRF security of kdf, and in distinguishing between the real-world AKE protocol and its ideal-world emulation of $\mathsf{SIM}_{\mathsf{AKE}}$ interacting with $\mathcal{F}_{\mathsf{khAKE}}$. Let $X' = Y = \{0,1\}^n$ be the domain and range of the ideal cipher IC used within IC^*, let X be the domain of (private,public) keys in AKE (e.g. for both 3DH and HMQV we have $X = \mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p), and let $\mathsf{map} : X \rightarrow \{0,1\}^n$ be ϵ_{map}-quasi-bijective.

Simulator construction. We split the description of simulator SIM into two phases: Figure 5.20 shows how SIM deals with creation and compromise of a password file and with adversary's ideal cipher queries, while Figure 4.19 shows how SIM deals with on-line sessions, i.e. how it executes AKE sessions and translates adversary's responses into on-line attacks on the aPAKE.

Simulator SIM uses as a sub-procedure the AKE-protocol simulator $\mathsf{SIM}_{\mathsf{AKE}}$, which exists by the assumption that the AKE protocol realizes functionality $\mathcal{F}_{\mathsf{khAKE}}$. Namely, SIM hands over to $\mathsf{SIM}_{\mathsf{AKE}}$ the simulation of all C-side and S-side AKE instances where parties run on honestly generated AKE keys. SIM employs $\mathsf{SIM}_{\mathsf{AKE}}$ to generate such keys, in password file initialization and in IC decryption queries, see Figure 5.20, and then it hands off to $\mathsf{SIM}_{\mathsf{AKE}}$ the handling of all AKE instances that run on such keys, see Figure 4.19. SIM cannot handle all AKE executions via $\mathsf{SIM}_{\mathsf{AKE}}$, because the adversary can guess client C's password pw

and form an envelope e' as IC encryption of arbitrary keys (a^*, B^*) under pw, in which case C executes AKE on adversarial keys (a^*, B^*). The ideal model of key-hiding AKE, i.e. functionality $\mathcal{F}_{\mathsf{khAKE}}$ of Figure 3.1, allows the environment to invoke AKE sessions on adversarially chosen counterparty public key, i.e. B^*, but it assumes that an honest party can use only its own previously generated key as its private key a. Since functionality $\mathcal{F}_{\mathsf{khAKE}}$ makes no claims for parties who run on inputs that violate this assumption, simulator SIM in this case simply executes the AKE protocol on behalf of C on such adversarially-chosen inputs (a^*, B^*). However, since this case implies a succesful on-line password guessing attack against client C, the simulation can give up on security on such sessions, hence w.l.o.g. this AKE execution could reveal inputs a^*, B^* to the adversary.

Game 0 *(real world)*: This is the interaction, shown in Figure 6.6, of environment $\mathcal{Z}$ with the real-world protocol KHAPE, except that the symmetric encryption scheme is idealized as an ideal cipher oracle. (Technically, this is a hybrid world where each party has access to the ideal cipher functionality IC.)



Initialize empty table $\mathsf{T_{IC}}$; (Notation $\mathsf{T}^{pw}_{\mathsf{IC}}.X'$ and $\mathsf{T}^{pw}_{\mathsf{IC}}.Y$ as in Fig. 5.20)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw^{\mathsf{uid}}_{\mathsf{S}})$ to S: Generate keys (a, A), (b, B), set $e^{\mathsf{uid}}_{\mathsf{S}} \leftarrow \mathsf{IC.E}(pw^{\mathsf{uid}}_{\mathsf{S}}, \mathsf{map}(a, B))$, and $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e^{\mathsf{uid}}_{\mathsf{S}}, b, A)$
- On new (pw, x') to $\mathsf{IC.E}$: Output $y \xleftarrow{r} Y \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.Y$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On new (pw, y) to $\mathsf{IC.D}$: Output $x' \xleftarrow{r} X' \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.X'$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Set $(e^{\mathsf{uid}}_{\mathsf{S}}, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, send $e^{\mathsf{uid}}_{\mathsf{S}}$ and start AKE session $\mathsf{S}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)$, set k_2 to $\mathsf{S}^{\mathsf{sid}}$ output;
  If $\mathcal{Z}$ sends $\tau' = \mathsf{kdf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set K_2, γ as $\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2)$, else as $\perp, \perp$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message e' to C: Set $(a, B) \leftarrow \mathsf{map}^{-1}(\mathsf{IC.D}(pw, e'))$, and start AKE session $\mathsf{C}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$, set k_1 to $\mathsf{C}^{\mathsf{sid}}$ output, send $\tau = \mathsf{kdf}(k_1, 1)$ to $\mathcal{Z}$;
  If $\mathcal{Z}$ sends $\gamma' = \mathsf{kdf}(k_1, 2)$ to $\mathsf{C}^{\mathsf{sid}}$, set $K_1 = \mathsf{kdf}(k_1, 0)$ else $K_1 = \perp$

Figure 3.17: Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE

GAME 1 *(embedding random keys in* $\mathsf{IC.D}$ *outputs)*: We modify processing of $\mathcal{Z}$'s query (pw, y) to $\mathsf{IC.D}$ for any $y \notin \mathsf{T}_{\mathsf{IC}}^{pw}.Y$, i.e. y for which $\mathsf{IC.D}(pw, y)$ has not been yet defined. On such query Game 1 generates fresh key pairs (a, A) and (b, B), sets $x' \xleftarrow{r} \mathsf{map}(a, B)$, and if $x' \notin \mathsf{T}_{\mathsf{IC}}^{pw}.X'$ then it sets $\mathsf{IC.D}(pw, y) \leftarrow x'$. If $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$, i.e. x' is already mapped by $\mathsf{IC.E}(pw, \cdot)$ to some value, Game 1 aborts. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then the game also sets $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, B)$.

The divergence this game introduces is due to the probability $(q_{\mathsf{IC}})^2/2^n$ of ever encountering an abort, and the statistical distance $q_{\mathsf{IC}}\epsilon_{\mathsf{map}}$ between random IC domain elements and images of map on random X elements, which leads to $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \le q_{\mathsf{IC}}\epsilon_{\mathsf{map}} + (q_{\mathsf{IC}})^2/2^n$.

GAME 2 *(random* $e_{\mathsf{S}}^{\mathsf{uid}}$ *in the password file)*: We change $\mathsf{StorePwdFile}$ processing by picking ciphertext $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random element in $\{0,1\}^n$ instead of via query to $\mathsf{IC.E}$, then we pick two key pairs (a, A), (b, B), define $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$, and sample $x' \xleftarrow{r} \mathsf{map}(a, B)$. If $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{T}_{\mathsf{IC}}^{pw}.Y$ for *any* pw, not necessarily $pw_{\mathsf{S}}^{\mathsf{uid}}$, the game aborts. The game also aborts if $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$. Otherwise the game sets $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow x'$ and $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$. The divergence this game introduces is due to the probability of abort occuring in either case, which leads to $|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \le q_{\mathsf{pw}}\epsilon_{\mathsf{map}} + 2q_{\mathsf{pw}}q_{\mathsf{IC}}/2^n$.

GAME 3 *(abort on ambiguous ciphertexts)*: In the ideal-world game simulator SIM identifies ciphertext e' which was output by the ideal cipher for some query (pw, x') to $\mathsf{IC.E}$, as an encryption of the *first* (pw, x') pair which satisfies this. To eliminate the possibility of ambiguous ciphertexts we introduce an abort if $\mathsf{IC.E}$ oracle picks the same ciphertext for any two queries (pw_1, x'_1) and (pw_2, x'_2). Since $\mathsf{IC.E}$ samples random outputs in Y we get $|\Pr[\mathsf{G3}] - \Pr[\mathsf{G2}]| \le (q_{\mathsf{IC}})^2/2^n$.

Taking stock of the game. Let us review how Game 3 operates: The initialization of password file $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ on password $pw_{\mathsf{S}}^{\mathsf{uid}}$ picks fresh keys (a, A), (b, B), picks $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random string, keeps the client and server public keys as $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) =$

(A, B), and programs $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ to $\mathsf{map}(a, B)$. Oracle $\mathsf{IC.D}$ on inputs (pw', y) for which decryption is undefined, picks fresh key pairs (a', A') and (b', B') and programs $\mathsf{IC.D}(pw', y)$ to $\mathsf{map}(a', B')$. In addition, if $y = e_{\mathsf{S}}^{\mathsf{uid}}$ then it assigns $pk_{\mathsf{S}}^{\mathsf{uid}}(pw') \leftarrow (a', B')$. Finally, encryption is now unambiguous, i.e. every ciphertext e can be output by $\mathsf{IC.E}$ on only one pair (pw, x').

This is already very close to how simulator SIM operates as well. The crucial difference between the ideal-world interaction and Game 3, is that in Game 3 keys $A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}$ are generated at the time of password file initialization, and $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set to $\mathsf{map}(a_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ at the same time. In the ideal-world game these keys are undefined until password compromise, and $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set only after offline dictionary attack succeeds in finding $pw_{\mathsf{S}}^{\mathsf{uid}}$. This delayed generation of the keys in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ is possible because AKE sessions which S and C run on these keys can be simulated without knowledge of these keys, an key-hiding AKE functionality allows precisely for such simulation, as we show next.

GAME 4 *(Using $\mathsf{SIM}_{\mathsf{AKE}}$ for AKE's on honestly-generated keys)*: In Game 4 we modify Game 3 by replacing all honest parties that run AKE instances on keys A, B generated either in password file initialization or by oracle $\mathsf{IC.D}$, with a simulation of these AKE instances via simulator $\mathsf{SIM}_{\mathsf{AKE}}$. Game 4 is shown in Figure 4.22. For notational brevity in Figure 4.22 we say that query (pw, x') to $\mathsf{IC.E}$ or (pw, y) to $\mathsf{IC.D}$ are new$^{(!)}$ as a shortcut for saying that table T_{IC} includes no prior tuple corresponding to these inputs, resp. $(pw, x', \cdot)$ and $(pw, \cdot, y)$. If such tuple exists then $\mathsf{IC.E}$ and $\mathsf{IC.D}$ oracles use the retrieved (key,input,output) tuple to answer the according query. We also omit the possibilities of the game aborts, because such aborts happen only with negligible probability. These aborts occur in three places, all marked $^{(*)}$: (1) When $e_{\mathsf{S}}^{\mathsf{uid}}$ is chosen in $\mathsf{StorePwdFile}$ the game aborts if $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{T}_{\mathsf{IC}}^{pw}.Y$ for any pw (not necessarily $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$); (2) When x' is then sampled as $x' \xleftarrow{r} \mathsf{map}(a, B)$, the game aborts if $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$; (3) When $x' \xleftarrow{r} \mathsf{map}(a, B)$ is sampled in $\mathsf{IC.D}$ query (pw, y) the game aborts also if $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$.

Initialize simulator $\mathsf{SIM_{AKE}}$, empty table $\mathsf{T_{IC}}$, and lists $CPK, PK_\mathsf{C}, PK_\mathsf{S}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid})$ to S: Initialize A and B via two Init calls to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ and $(\mathsf{Compromise}, B)$ to $\mathsf{SIM_{AKE}}$, define a and b as $\mathsf{SIM_{AKE}}$'s responses, add A to PK_C, B to PK_S, and both to CPK, pick$^{(*)}$ $e_\mathsf{S}^\mathsf{uid} \xleftarrow{r} Y$, set$^{(*)}$ $x' \xleftarrow{r} \mathsf{map}(a, B)$, add $(pw_\mathsf{S}^\mathsf{uid}, x', e_\mathsf{S}^\mathsf{uid})$ to $\mathsf{T_{IC}}$, set $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_\mathsf{S}^\mathsf{uid}, b, A)$ and $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid}) \leftarrow (A, B)$
- On new$^{(!)}$ (pw, x') to $\mathsf{IC.E}$: Output $y \xleftarrow{r} Y \setminus \mathsf{T}_\mathsf{IC}^{pw}.Y$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$: Initialize A and B via two Init calls to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK and PK_C, add B to PK_S, set$^{(*)}$ $x' \xleftarrow{r} \mathsf{map}(a, B)$ add (pw, x', y) to $\mathsf{T_{IC}}$, output x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_\mathsf{S}^\mathsf{sid}$, set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{hbc}$, output $e_\mathsf{S}^\mathsf{uid}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM_{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_\mathsf{C}^\mathsf{sid}$ and:
  1. If $e' = e_\mathsf{S}^\mathsf{uid}$, set $x' \leftarrow \mathsf{IC.D}(pw, e_\mathsf{S}^\mathsf{uid})$, $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{hbc}(g^a, B)$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM_{AKE}}$
  2. If $e' \neq e_\mathsf{S}^\mathsf{uid}$, check if e' was output by $\mathsf{IC.E}$ on (pw, x') for some x' and:
     (a) If not, set $x' \leftarrow \mathsf{IC.D}(pw, e')$, $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{hbc}(g^a, B)$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM_{AKE}}$
     (b) If so, set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, run C^sid of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If C^sid terminates with k, output $\tau \leftarrow \mathsf{kdf}(k, 1)$ and $K_1 \leftarrow \mathsf{kdf}(k, 0)$

Responding to AKE messages:

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(A, B)$ then change it to $\mathsf{act}(A, B)$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{act}(A, B)$ set $k_1 \leftarrow R_\mathsf{C}^\mathsf{sid}(A, B, \alpha)$
  2. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(A, B)$: If $(A, B) = (A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ and S^sid outputted key k_2 then copy this k_2 to k_1, otherwise pick $k_1 \xleftarrow{r} \{0,1\}^\kappa$

  Output $\tau \leftarrow \mathsf{kdf}(k_1, 1)$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to S^sid:
  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}$, set $k_2 \leftarrow R_\mathsf{S}^\mathsf{sid}(B_\mathsf{S}^\mathsf{uid}, A_\mathsf{S}^\mathsf{uid}, \alpha)$
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$: If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ and C^sid outputted key k_1 then copy this k_1 to k_2, otherwise pick $k_2 \xleftarrow{r} \{0,1\}^\kappa$

  If $\tau' = \mathsf{kdf}(k_2, 1)$ output $(K_2, \gamma) \leftarrow (\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2))$, else $(K_2, \gamma) \leftarrow (\perp, \perp)$
- On γ' to C^sid: If $\gamma' = \mathsf{kdf}(k_1, 2)$ output $K_1 \leftarrow \mathsf{kdf}(k_1, 0)$ else $K_1 \leftarrow \perp$
- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_\mathsf{P}^\mathsf{sid}(pk, pk', \alpha)$ if $pk \in PK_\mathsf{P}$, $pk' \in CPK$

Figure 3.18: Proof of KHAPE security: Game 4

Game 4 operates like Game 3, except that it outsources AKE key generation in $\mathsf{StorePwdFile}$ and $\mathsf{IC.D}$ to $\mathsf{SIM}_{\mathsf{AKE}}$, and whenever $\mathsf{S}^{\mathsf{sid}}$ or $\mathsf{C}^{\mathsf{sid}}$ runs AKE on such keys these executions are outsourced to $\mathsf{SIM}_{\mathsf{AKE}}$, while the game emulates what $\mathcal{F}_{\mathsf{khAKE}}$ would do in response to $\mathsf{SIM}_{\mathsf{AKE}}$'s actions. In particular, Game 4 initializes random function $R^{\mathsf{sid}}_{\mathsf{P}}$ for every AKE session $\mathsf{P}^{\mathsf{sid}}$ invoked by emulated $\mathcal{F}_{\mathsf{khAKE}}$. Whenever C and S run an AKE instance under keys generated by AKE key generation the game, playing $\mathcal{F}_{\mathsf{khAKE}}$, triggers $\mathsf{SIM}_{\mathsf{AKE}}$ with messages resp. $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$. When $\mathsf{SIM}_{\mathsf{AKE}}$ translates the real-world adversary's behavior into $\mathsf{Interfere}$ actions on these sessions, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by marking these sessions as actively attacked. If $\mathsf{SIM}_{\mathsf{AKE}}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ on activey attacked session, its output key k is set to $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the keys this session runs under, which are $(B^{\mathsf{uid}}_{\mathsf{S}}, A^{\mathsf{uid}}_{\mathsf{S}})$ for S, and keys (A, B) defined by $\mathsf{IC.D}(pw, e')$ for C. The game must also emulate $\mathsf{ComputeKey}$ interface of $\mathcal{F}_{\mathsf{khAKE}}$ and let $\mathsf{SIM}_{\mathsf{AKE}}$ evaluate $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ for any $pk \in PK_{\mathsf{P}}$ and any $pk' \in CPK$. (Note that all sessions emulated by $\mathsf{SIM}_{\mathsf{AKE}}$ run on public keys pk' which are created by the Init interface.) Set PK_{S} contains only one key, $B^{\mathsf{uid}}_{\mathsf{S}}$, while set PK_{C} contains $A^{\mathsf{uid}}_{\mathsf{S}}$ and all keys A' created by $\mathsf{IC.D}$ queries. Set CPK consists of $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$, because these were compromised in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ initialization, which used the corresponding private keys, and all client-side keys A' generated in $\mathsf{IC.D}$ queries, because each $\mathsf{IC.D}$ query creates and immediately compromises key A', since it needs to embed the corresponding private key a' into $\mathsf{IC.D}$ output. Finally, if $\mathsf{SIM}_{\mathsf{AKE}}$ sends NewKey on non-attacked session, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by issuing random keys to such sessions except if $\mathsf{C}^{\mathsf{sid}}$ runs under key pair $(A', B') = (A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, which matches the key pair used by $\mathsf{S}^{\mathsf{sid}}$, in which case the game copies the key output by the session which terminates first into the key output by the session which terminates second. The rest of the code is as in Game 3: C uses its key k_1 to compute authenticator $\tau = \mathsf{kdf}(k_1, 1)$ and its local output $K_1 = \mathsf{kdf}(k_1, 0)$, while S uses its key k_2 to verify the incoming authenticator τ' and outputs $K_2 = \mathsf{kdf}(k_2, 0)$ if $\tau' = \mathsf{kdf}(k_2, 1)$ and $K_2 = \perp$ otherwise.

The one case where a party might not run AKE on keys generated via a call to $\mathsf{SIM}_{\mathsf{AKE}}$

is client session C which receives e' which was output by $\mathsf{IC.E}(pw, x')$ for some x' and pw matching the password input to $\mathsf{C}^{\mathsf{sid}}$. In this case $\mathsf{C}^{\mathsf{sid}}$ runs AKE on $(a, B) = \mathsf{map}^{-1}(x')$, and since wlog these keys are chosen by the adversary and not by $\mathsf{SIM}_{\mathsf{AKE}}$, we cannot outsource that execution to $\mathsf{SIM}_{\mathsf{AKE}}$. As we said above, functionality $\mathcal{F}_{\mathsf{khAKE}}$ does not admit honest parties running AKE on arbitrary private keys a, hence $\mathsf{SIM}_{\mathsf{AKE}}$ does not have an interface to simulate such executions. In Game 4 such AKE instances are executed as in Game 3: This is the case in step (2b) in Figure 4.22.

Since Game 4 and Game 3 are identical except for replacing real-world AKE executions with the game emulating functionality $\mathcal{F}_{\mathsf{khAKE}}$ interacting with $\mathsf{SIM}_{\mathsf{AKE}}$, it follows that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$

Game 5 *(delay $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ generation until password compromise)*: In Game 4 keys $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ are initialized and compromised in $\mathsf{StorePwdFile}$, in Game 5 we postpone these steps until password compromise. This change can be done in several steps.

Denote first step as Game 5(a), we remove compromising $B^{\mathsf{uid}}_{\mathsf{S}}$, adding it to CPK and setting $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$. $\mathcal{Z}$ cannot notice this change because in Game 4, only $\mathsf{StealPwdFile}$ will need $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$, and compromising $B^{\mathsf{uid}}_{\mathsf{S}}$ to get $b^{\mathsf{uid}}_{\mathsf{S}}$ is not needed anywhere else except when generating $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$.

In Game 5(b) we make a change in $\mathsf{IC.D}$, that if $y \neq e^{\mathsf{uid}}_{\mathsf{S}}$ then set $x' \xleftarrow{r} X' \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.X'$, while in Game 4 we set $x' \xleftarrow{r} \mathsf{map}(a, B)$ for randomly initialized (a, B), with restriction that this x' hasn't been mapped before. The divergence this change introduces is due to the statistical distance $q_{\mathsf{IC}}\epsilon_{\mathsf{map}}$ between random IC domain elements and images of map on random X elements.

Then in Game 5(c) we remove compromising $A^{\mathsf{uid}}_{\mathsf{S}}$, adding it to CPK, setting x' and adding $(pw^{\mathsf{uid}}_{\mathsf{S}}, x', e^{\mathsf{uid}}_{\mathsf{S}})$ to T_{IC} in $\mathsf{StorePwdFile}$, and delay them to new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$. After this change, in $\mathsf{StorePwdFile}$ we now only initialize $(A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, add them to PK and pick $e^{\mathsf{uid}}_{\mathsf{S}}$. Since $(pw^{\mathsf{uid}}_{\mathsf{S}}, x', e^{\mathsf{uid}}_{\mathsf{S}})$ is no longer added to T_{IC} in $\mathsf{StorePwdFile}$, query $(pw^{\mathsf{uid}}_{\mathsf{S}}, e^{\mathsf{uid}}_{\mathsf{S}})$ is now

Initialize simulator $\mathsf{SIM_{AKE}}$, empty table $\mathsf{T_{IC}}$, and lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Pick $e_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} Y$, mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ as fresh
- On new$^{(!)}$ (pw, x') to $\mathsf{IC.E}$: Output $y \xleftarrow{r} Y \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.Y$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \xleftarrow{r} X' \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is fresh or $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$ then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, initialize A and B via Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_{C} and B to PK_{S}
     (b) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$ and $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$ set $(A, B) \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     In both cases (a) and (b), set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, B)$, define a as $\mathsf{SIM_{AKE}}$'s response to $(\mathsf{Compromise}, A)$, add A to CPK, and set $x' \xleftarrow{r} \mathsf{map}(a, B)$

  Add (pw, x', y) to $\mathsf{T_{IC}}$ and send back x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ $\mathsf{compromised}$ and: If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$ then set $(A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$; Else initialize A and B via Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_{C} and B to PK_{S}; In either case, set $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$, define b as $\mathsf{SIM_{AKE}}$'s response to $(\mathsf{Compromise}, B)$, add B to CPK, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_{\mathsf{S}}^{\mathsf{sid}}$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, output $e_{\mathsf{S}}^{\mathsf{uid}}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM_{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and:
  1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ then: (1) set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, otherwise set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$; (2) send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM_{AKE}}$
  2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ then:
     (a) If e' was not output by $\mathsf{IC.E}$ or it was output on (pw', x') for $pw' \neq pw$, then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM_{AKE}}$
     (b) If e' was output by $\mathsf{IC.E}$ on (pw, x') then set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, run $\mathsf{C}^{\mathsf{sid}}$ of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If $\mathsf{C}^{\mathsf{sid}}$ terminates with k, output $\tau \leftarrow \mathsf{kdf}(k, 1)$ and $K_1 \leftarrow \mathsf{kdf}(k, 0)$

<u>Responding to AKE messages:</u>

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$, otherwise $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ set $k_1 \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}, \alpha)$, output $\tau \leftarrow \mathsf{kdf}(k_1, 1)$
  2. Otherwise output $\tau \xleftarrow{r} \{0,1\}^{\kappa}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\tau' = \mathsf{kdf}(k_2, 1)$ for $k_2 = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, \alpha)$ where $(A, B) = pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$, then output $(K_2, \gamma) \leftarrow (\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2))$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by $\mathsf{C}^{\mathsf{sid}}$ where $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, then output $K_2 \xleftarrow{r} \{0,1\}^{\kappa}$ and $\gamma \xleftarrow{r} \{0,1\}^{\kappa}$
  3. In all other cases output $(K_2, \gamma) \leftarrow (\perp, \perp)$
- On γ' to $\mathsf{C}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ and $\gamma' = \mathsf{kdf}(k_1, 2)$, output $K_1 \leftarrow \mathsf{kdf}(k_1, 0))$
  2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ and γ' was generated by $\mathsf{S}^{\mathsf{sid}}$ for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, output K_1 equal to the key K_2 output by $\mathsf{S}^{\mathsf{sid}}$
  3. In all other cases output $K_1 \leftarrow \perp$
- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ if $pk \in PK_{\mathsf{P}}$, $pk' \in CPK$

Figure 3.19: KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 8)

new$^{(!)}$ to $\mathsf{IC.D}$, and we add that in this case $\mathsf{IC.D}$ responds by retrieving $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$, compromising $A_\mathsf{S}^\mathsf{uid}$, setting corresponding x' and adding $(pw_\mathsf{S}^\mathsf{uid}, x', e_\mathsf{S}^\mathsf{uid})$ to $\mathsf{T_{IC}}$. For any other queries, $\mathsf{IC.D}$ reacts same as in Game 5(b). Game 5(c) and Game 5(b) is identical since we only postpone executing those steps removed from $\mathsf{StorePwdFile}$.

In Game 5(d) we further remove usage of $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ when responding to AKE messages, except for input to $R_\mathsf{P}^\mathsf{sid}$ in actively attacked sessions. We change $\mathsf{hbc}(A, B)$ in Game 5(c) to $\mathsf{hbc}_\mathsf{S}^\mathsf{uid}$ if $(A, B) = (A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$, and rnd otherwise. Similarly we change $\mathsf{act}(A, B)$ in Game 5(c) to $\mathsf{act}_\mathsf{S}^\mathsf{uid}$ if $(A, B) = (A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$, which corresponds to active attack, otherwise set to rnd and derive corresponding k_1 from random element of $\{0, 1\}^\kappa$ instead of $R_\mathsf{C}^\mathsf{sid}(A, B, \alpha)$, from randomness of $R_\mathsf{C}^\mathsf{sid}$ this change makes indistinguishable difference to $\mathcal{Z}$. Since these are only notational changes and $\mathcal{Z}$ cannot notice them, Game 5(d) and Game 5(c) are identical to $\mathcal{Z}$.

Finally, in Game 5(e) we remove steps of initializing $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ via $\mathsf{SIM_{AKE}}$ in $\mathsf{StorePwdFile}$ and delay them to $\mathsf{StealPwdFile}$ or $\mathsf{IC.D}(pw_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})$, depending on which happens first. In order to set $\mathsf{IC.D}(pw_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})$ only after $\mathcal{A}$ finds $pw_\mathsf{S}^\mathsf{uid}$ via successful offline dictionary attack, we first mark $pw_\mathsf{S}^\mathsf{uid}$ fresh in $\mathsf{StorePwdFile}$, and mark it $\mathsf{compromised}$ anytime $\mathcal{A}$ runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$.

If $\mathcal{A}$ first runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, we initialize $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ via Init calls to $\mathsf{SIM_{AKE}}$, add $A_\mathsf{S}^\mathsf{uid}$ to PK_C and $B_\mathsf{S}^\mathsf{uid}$ to PK_S, and later upon query $\mathsf{IC.D}(pw_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})$, if $pw_\mathsf{S}^\mathsf{uid}$ is already marked $\mathsf{compromised}$, we simply retrieve $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$, then compromise $A_\mathsf{S}^\mathsf{uid}$ and set x' as in Game 5(d). In the other case, if $\mathsf{IC.D}(pw_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})$ runs first, which means at this moment $pw_\mathsf{S}^\mathsf{uid}$ must be fresh, we treat it same way as before, and just like any other $pw \neq pw_\mathsf{S}^\mathsf{uid}$, where we init $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ via $\mathsf{SIM_{AKE}}$, add them to PK and save $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ into $pk_\mathsf{S}^\mathsf{uid}(pw_\mathsf{S}^\mathsf{uid})$ for future retrival. We also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid} \rangle$, and later if $\mathcal{A}$ runs $\mathsf{StealPwdFile}$ and there exists record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid} \rangle$, then just directly retrieve $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ from $pk_\mathsf{S}^\mathsf{uid}(pw_\mathsf{S}^\mathsf{uid})$ and skip initialization. In addition we also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ upon query $\mathsf{IC.D}(pw, e_\mathsf{S}^\mathsf{uid})$ even if $pw \neq pw_\mathsf{S}^\mathsf{uid}$. Game 5(e) is identical to Game 5(d) since we only postpone $(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$ initialization. Thus we conclude: $|\Pr[\mathsf{G5}] - \Pr[\mathsf{G4}]| \leq q_\mathsf{IC}\epsilon_\mathsf{map}$

Game 6 *(replace* kdf *output with random string in passive sessions)*: In Game 5, in passive sessions, i.e. any sessions except actively attacked sessions, τ, γ are all derived from kdf of k_1 or k_2. In Game 6 in these sessions we remove usage of kdf and directly assign random elements of $\{0,1\}^\kappa$ to these values. Also we replace verifying τ', γ' via checking $\tau' = \mathsf{kdf}(k_2, 1), \gamma' = \mathsf{kdf}(k_1, 2)$ with checking whether they're generated by corresponding hbc parties, since these two checking methods are actually equal. In addition, we further remove usage of k_1 and k_2 in passive sessions, and instead set $K_2 \leftarrow \{0,1\}^\kappa$, and in matching sessions we copy K_2 to K_1, as Game 5 copy k_1 to k_2 or vice versa in such sessions. Since there're at most q_{ses} such sessions, and from security of kdf, the difference between Game 5 and Game 6 is negligible to $\mathcal{Z}$, i.e. $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}} \epsilon^{\mathcal{Z}}_{\text{kdf}}(\mathsf{SIM}_{\mathsf{AKE}})$

Game 7 *(Ideal-world game)*: This is the ideal-world interaction, i.e. an interaction of environment $\mathcal{Z}$ with simulator SIM and functionality $\mathcal{F}_{\mathsf{aPAKE}}$, shown in Figure 6.11.

Observe that Game 6 is identical to the ideal-world Game 8. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 4.4.

3.7 Concrete aPAKE Instantiation: KHAPE-HMQV

We include a concrete aPAKE protocol we call KHAPE-HMQV, which results from instantiating protocol KHAPE shown in Section 6.2 with HMQV as the key-hiding AKE (as proved in Section 4.2.2). The resulting protocol is shown in Figure 3.20. It uses only 1 fixed-base exponentiation plus 1 variable-base (multi)exponentiation for each party, and 1 ideal cipher decryption for the client. It has 3 flows if the server initiates and 4 if the client initiates. The communication costs include one group element and a κ-bit key authenticator for both sides plus an ideal cipher encryption of a field element a and another group element B from server

• global hash functions $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, $\mathsf{H}' : \{0,1\}^* \rightarrow \mathbb{Z}_p$
• group $\mathbb{G}$ of prime order p with generator g
• cipher $(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ on space $\mathbb{Z}_p \times \mathbb{G}$ (see also page 67)

Password File Initialization on S's input $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$:

S picks two fresh AKE keys (a, A) and (b, B), sets $e \leftarrow \mathsf{IC}^*.\mathsf{E}(pw, (a, B))$
S stores $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e, b, A)$ and discards all other ephemeral values

C on $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$		S on $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$
$x \xleftarrow{r} \mathbb{Z}_p$, $X \leftarrow g^x$		$y \xleftarrow{r} \mathbb{Z}_p$, $Y \leftarrow g^y$
$(a, B) \leftarrow \mathsf{IC}^*.\mathsf{D}(pw, e)$	$\xleftarrow{e,\ Y}$	$(e, b, A) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$
$d_\mathsf{C} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X)$		
$e_\mathsf{C} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$		
$\sigma_\mathsf{C} \leftarrow (Y \cdot B^{e_\mathsf{C}})^{x + d_\mathsf{C} \cdot a}$		
$k_1 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma_\mathsf{C})$		
$\tau \leftarrow \mathsf{kdf}(k_1, 1)$	$\xrightarrow{\tau,\ X}$	$d_\mathsf{S} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X)$
		$e_\mathsf{S} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$
		$\sigma_\mathsf{S} \leftarrow (X \cdot A^{d_\mathsf{S}})^{y + e_\mathsf{S} \cdot b}$
		$k_2 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma_\mathsf{S})$
		$\gamma \leftarrow \perp$ if $\tau \neq \mathsf{kdf}(k_2, 1)$
	$\xleftarrow{\gamma}$	else $\gamma \leftarrow \mathsf{kdf}(k_2, 2)$
$K_1 \leftarrow \perp$ if $\gamma \neq \mathsf{kdf}(k_1, 2)$		$K_2 \leftarrow \perp$ if $\tau \neq \mathsf{kdf}(k_2, 1)$
else $K_1 \leftarrow \mathsf{kdf}(k_1, 0)$		else $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$
output K_1		output K_2

Figure 3.20: KHAPE with HMQV: Concrete aPAKE protocol KHAPE-HMQV

to client. Implementations of an ideal cipher over field elements may expand the ciphertext by $\Omega(\kappa)$ bits and require a hash-to-curve operation, see Sec. 3.8.

While we are showing the protocol with the encryption of credentials done on the server side during password registration (initialization), this can be done interactively by the server sending its public key and the user encrypting it together with its private key under the password (or it can all be done on the client side if the client chooses the server's public key). It is important to highlight that the server needs a random independent pair of private-public keys per user. One optimization is to omit the encryption of the user's private key,

and instead derive this key from the password. Our analysis can be adapted to this case.

We note that KHAPE can be made into a Strong aPAKE (saPAKE), secure against pre-computation attacks, using the technique of [88]. Namely, running an OPRF protocol on pw between client and server and deriving the credential encryption key from the output of the OPRF. In addition to providing saPAKE security, the OPRF strengthens the protocol against online client-side attacks (the attacker cannot have a pre-computed list of passwords to try) and it allows for distributing the server through a threshold OPRF. As discussed in the introduction, the break of the OPRF in the context of KHAPE voids the above benefits but does not endanger the password (a major advantage of KHAPE over OPAQUE).

3.8 Curve Encodings and Ideal Cipher

3.8.1 Quasi bijections

Protocol KHAPE encrypts group elements (server's public key pk_S) using an encryption function modeled as an ideal cipher which works over a space $\{0,1\}^n$ for some n. Thus, prior to encryption, group elements need to be encoded as bitstrings of length n to which the ideal cipher will be applied. We require such encoding, denoted map, from G to $\{0,1\}^n$ to be a bijection (or close to it) so that if e is an encryption of $g \in G$ under password pw, its decryption under a different pw' returns a random element in G. The following definition considers randomized encodings.

Definition 3.3. *A randomized ε-quasi bijection* map *with domain A, randomness space $R = \{0,1\}^\rho$ and range B consists of two algorithms* map *and* map^{-1}, $\mathsf{map} : A \times R \to B$ *and* $\mathsf{map}^{-1} : B \to A$ *with the following properties:*

1. map^{-1} *is deterministic and for all $a \in A, r \in R, \mathsf{map}^{-1}(\mathsf{map}(a,r)) = a$;*

2. map *maps the uniform distribution on $A \times R$ to a distribution on B that is ε-close to uniform.*

The term ε-close refers to a statistical distance of at most ε between the two distributions. It can also be used in the sense of computational indistinguishability, e.g., if implementing randomness using a PRG. To accommodate bijections whose randomized map from A to B may exceed a given time bound in some inputs, one can consider the range of map to include an additional element $\perp$ to which such inputs are mapped. A simpler way is to define that such inputs are mapped to a fixed element in B. The probability of inputs mapped to that value is already accounted for in the statistical distance bound ε. We use *quasi bijection* without specifying ε when we assume this value to be negligible.

Quasi bijections from field elements to bitstrings. We are interested in quasi-bijective encoding into the set $\{0,1\}^n$ over which the IC encryption works. Most mappings presented below have a field $\mathbb{Z}_q$ as the range, in which case a further transformation (preserving quasi-bijectiveness) may be needed. Note that when representing elements of $\mathbb{Z}_q$ as n-bit numbers for $n = \lceil \log\ q \rceil$, the uniform distribution on $\mathbb{Z}_q$ is ε-close to the uniform distribution over $\{0,1\}^n$ for $\varepsilon = (2^n \bmod q)/q$. So when q is very close to 2^n, one can use the bit representation of field elements directly, and this is the case for many of the standardized elliptic curves. When this is not the case, one maps $u \in \mathbb{Z}_q$ to a $(n+k)$-bit integer selected as $u + tq$ for t randomly chosen as a non-negative integer $< (2^{n+k} - u)/q$. The resulting distribution is 2^{-k}-close to the uniform distribution over $\{0,1\}^{n+k}$.

3.8.2 Implementing quasi-bijective encodings

We focus on the case where G is an elliptic curve. There is a large variety of well-studied quasi-bijective encodings in the literature (cf. [117, 45, 66, 32, 121]). We survey some representative examples for elliptic curve groups $EC(q)$ over fields of large prime-order q.

Note that we use both directions of these encodings in KHAPE: From pk_S to a bitstring when encrypting pk_S at the time of password registration, and from a bitstring to a curve point when the client decrypts pk_S. This means that the performance of the latter operation is more significant for the efficiency of the protocol. Fortunately this is always the more efficient direction, even though the other direction is quite efficient too for the maps discussed below.

Elligator-squared [121, 96]. This method applies to most elliptic curves and accommodates ε-quasi bijections for the *whole set of curve points* with negligible values of ε.

Curve points are encoded as a pair of field elements $(u, v) \in \mathbb{Z}_q^2$. There is a deterministic function f from $\mathbb{Z}_q$ to EC such that $P \in EC$ is represented by (u, v) if and only if $P = f(u) + f(v)$. Given a point P there is a randomized procedure R_f that returns such encoding (u, v).

In [121] (Theorem 1), it is proven that for suitable choices of f, R_f is an ε-quasi bijection into $(\mathbb{Z}_q)^2$, with $\varepsilon = O(q^{-1/2})$ (see Definition 3.3). Since u, v are field elements, a further bijection into bitstrings may be needed as specified in Section 3.8.1.

In [96], the above construction is improved by allowing both u and v to be represented directly as bit strings: u as a string of $\lfloor q \rfloor$ bits and v can be be shortened even further (the amount of shortening increases the statistical distance for the quasi bijection from EC to the distribution of bitstrings (u, v)). This encoding uses two functions f, g where a point P is recovered from (u, v) as $P = f(u) + g(v)$ (in this case, function g can be simply $g(v) = v \cdot P$).

The performance of Elligator-squared depends on the functions f, g whose cost with typical instantiations (e.g., Elligator, SWU) is dominated by a single base-field exponentiation at the cost of a fraction ($\approx$10-15%) of a scalar multiplication. Implementing $g(v) = v \cdot P$ is also a low-cost option (also allowing to shorten v [96]). The cost of the inverse map, from a curve point to its bitstring encoding, for the curves analyzed in [121] is 3 base-field exponentiations.

Elligator2. This mapping from [32] is of more restricted applicability than Elligator-squared as it applies to a smaller set of curves (e.g., it requires an element of order 2). Yet, this class includes some of the common curves used in practice, particularly Curve25519. Eligator2 defines an injective mapping between the integers $\{0, \ldots, (q-1)/2\}$ and (about) half of the elements in the curve. To be used in our setting, it means that when generating a pair $(sk_S, pk_S{=}g^{K_S})$ for the server during password registration, the key generation procedure will choose a random K_S and will test if the resultant pk_S has a valid encoding under Elligator2. If so, it will keep this pair, otherwise it will choose another random pair and repeat until a representable point is found. The expected number of trials is 2 and the testing procedure is very efficient (and only used during registration, not for login).

The advantages of Ellligator2 include the use of a single field element as a point representation (which requires further expansion into a bit string only if q is not close to 2^n) and the map is injective, hence quasi-bijective with $\varepsilon = 0$ over the subset of encodable curve elements. Both directions of the map are very efficient, costing about a single base-field exponentiation (a fraction of the cost of a scalar multiplication).

Detailed implementation information for the components of the above transforms is found in [64, 32, 122]. See [18] for some comparison between Elligator2 and Elligator-squared.

3.8.3 Ideal Cipher Constructions

Protocol KHAPE uses an ideal cipher to encrypt group elements, specifically a pair $(K_{\mathsf{C}}, pk_{\mathsf{S}})$ where both elements are encoded as bitstrings to fit the ideal cipher interface as described in previous subsections.

Thus, we consider the input to the encryption simply as a bitstring of a given fixed length, and require implementations of ideal ciphers of sufficiently long block length. For example,

the combined input length for curves of 256 bits ranges between 512 and 1024 bits. Constructions of encryption schemes that are indifferentiable from an ideal cipher have been investigated extensively in the literature. Techniques include domain extension mechanisms (e.g., to expand the block size for block ciphers, including AES) [50], Feistel networks and constructions from random oracles [55, 82, 52], dedicated constructions such as those based on iterated Even-Mansour and key alternating ciphers [54, 17, 61, 61], and basic components such as wide-input (public) random permutations [35, 34, 53]. A recent technique by McQuoid et al. [109], builds a dedicated transform that can replace the ideal cipher in cases where encryption is "one-time", namely, keys (or cipher instances) are used to encrypt a single message (as in our protocols). They build a very efficient transform using a random oracle with just two Feistel rounds.

In Chapter 5 we show a new efficient Ideal Cipher construction which we called Half-Ideal Cipher, and we show a dedicated analysis for the use of this technique in our context.

Chapter 4

OKAPE:Asymmetric PAKE with low computation and communication

4.1 Introduction

The work of KHAPE [74] considered minimal-cost aPAKE's, and showed an aPAKE protocol which nearly matches the computational cost of unauthenticated key exchange (KE), namely Diffie-Hellman (uDH), which is 1fb+1vb exp per party (i.e., 1 fixed-base and 1 variable-base exponentiation). The KE cost is a lower-bound for both PAKE and aPAKE because aPAKE $\Rightarrow$ PAKE $\Rightarrow$ KE. However, the minimal-cost aPAKE protocol of [74] is not close to KE in round complexity. Indeed, the aPAKE of [74] takes 3 rounds assuming the server initiates the protocol, while uDH takes a single simultaneous flow, where each party sends a single protocol message without waiting for the counterparty. Note that this minimal round complexity is achieved by minimal-cost universally composable (UC) PAKE's, including EKE [28, 26, 109], SPAKE2 [10, 4], and TBPEKE [113, 4].[1]

[1] Abdalla et al. [4] show that SPAKE2 [10] and TBPEKE [113] realize a relaxed version of the UC PAKE functionality of Canetti et al. [48].

Our Contributions. We show that cost-optimal aPAKE does not have to come at the expense of round complexity. We do so with a new aPAKE construction, called OKAPE which is a generic compiler that construct aPAKE's from any key-hiding *one-time-key* Authenticated Key Exchange (*otkAKE*). The construction uses the Random Oracle Model (ROM) and an Ideal Cipher (IC) on message spaces formed by otkAKE public keys. We define the notion of key-hiding otkAKE as a relaxation of the UC key-hiding AKE of [74], and we show that it is realized by "one-pass" variants of 3DH and HMQV which were shown as UC key-hiding AKEs in [74].

The compiler instantiated with one-pass HMQV produce a concrete aPAKE schemes which we call OKAPE-HMQV. It has close to optimal computational cost of 1fb+1vb exp for the client and 1fb+1mvb exp for the server, where mvb stands for multi-exponentiation with two bases.

Protocol OKAPE requires 2 communication rounds if the server initiates the protocol, and 3 if the client does. OKAPE supports *(publicly) salted* password hashes, which have several security and operational benefits over unsalted ones (see Note 1 below). Note that every aPAKE can be generically transformed to support a publicly salted hash if the server first sends the salt to the client and the two parties run aPAKE on the password appended by the salt. However, among prior UC aPAKE's that use unsalted password hashes [72, 92, 84, 119], only the aPAKE of Jutla and Roy [92] and Hwang et al. [84] match the round complexity of OKAPE-HMQV after this transformation, but they do not match its computational cost: The PAKE-to-aPAKE compiler of [84] instantiatied with a minimal-cost PAKE has a total computational cost of 3fb+3vb exps, i.e. 50% more than uDH, while the aPAKE of [92] is significantly more expensive, in particular because it uses bilinear maps.

The only prior UC aPAKE's that natively support salted hashes with 3 or fewer communication rounds is the 3-round protocol OPAQUE of Jarecki et al. [88, 89] and the 2-round CKEM-based protocol of Bradley et al. [44]. Both of these protocols have at least 2 times

scheme	client$^{(1)}$	server$^{(1)}$	rounds$^{(2)}$	salting	EA$^{(3)}$	assump.	model
Jutla-Roy[92]	O(1)	O(1)	1	none	none	XDH	RO
KC-SPAKE2+[119]	2f+2v	2f+2v	3(C)	none	C+S	CDH	RO
OKAPE-HMQV [*]	1f+1.2v	1f+1.2v	2(S)	public	S	gapDH	RO/IC
Hwang[84] +EKE[28]	2f+1v	1f+2.2v	2(S)	public	S	CDH	RO/IC
KHAPE-HMQV[74]	1f+1.2v	1f+1.2v	3(S)	public	C+S	gapDH	RO/IC
CKEM-saPAKE[44]	10f+1v	2f+2v	2(C)	private	C	sDH,DDH	RO
OPAQUE-HMQV[89]	2f+2.2v	1f+2.2v	3(C)	private	C+S	OM-DH	RO

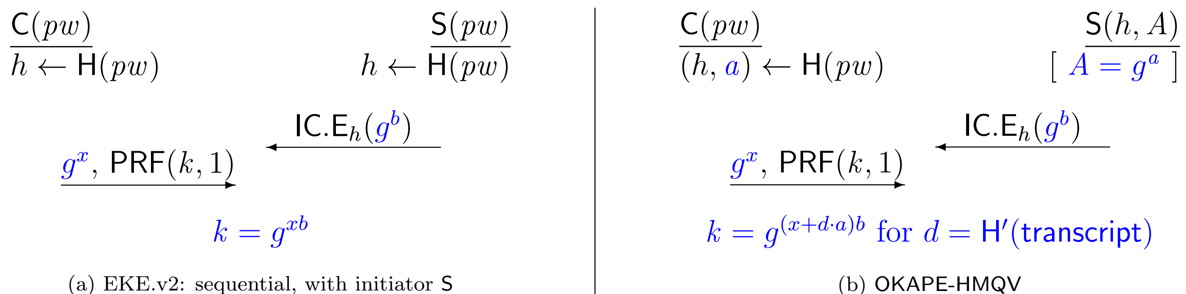
Table 4.1: Comparison of UC aPAKE schemes, with our schemes marked [*]: (1) f,v denote resp. fixed-base and variable-base exponentiation (expo), two-base multi-expo is counted as 1.2v, O(1) stands for significantly larger costs including bilinear maps; (2) x(C) and x(S) denote x rounds if respectively client starts or server starts, while "1" denotes a single-flow protocol; (3) EA column lists the parties that explicitly authenticate their counterparty at protocol termination. OPAQUE-HMQV appeared in [88], but above we give optimized performances characteristics due to [89].

higher computational costs than uDH. However, both [88] and [44] provide *strong* aPAKEs (saPAKE), where the salt in the password hash is private, whereas OKAPE supports publicly salted hash, see Note 1 below. In table 5.6 we compare efficiency and security properties of prior UC aPAKE's and the concrete protocols we propose. Note that all schemes which achieve explicit authentication for only one party can also achieve it for the other using one additional key confirmation flow. Note also that any single-flow aPAKE can be transformed so it achieves explicit authentication for both parties in 3 flows, regardless of which party starts. In the table we do not include aPAKE schemes which were not proven in UC models so far, including VPAKE [31] or PAK-X [41], but both schemes are slightly costlier than e.g. KC-SPAKE2+ [119], see e.g. [44] for exact cost comparisons.

Main Idea: Encrypted Key Exchange paradigm for aPAKE. Our protocols are compilers which build aPAKE's from any key-hiding otkAKE, i.e. an AKE where one party uses a one-time key. In both protocols server S picks a one-time public key pair (b, B) and sends the public key B encrypted under a password hash h to client C, who decrypts

it under a hash of its password pw. C also has a long-term private key a derived as a password hash as well, i.e. $(h, a) = \mathsf{H}(pw)$, and S holds the corresponding public key A together with h in the password file for this client. The two parties then run a *key-hiding* otkAKE on respective inputs (a, B) and (b, A), but here the two compilers diverge: In OKAPE the otkAKE subprotocol is executed in a black-box way, and it is followed by explicit key confirmation message from C to S. The protocol is shown secure if password-encryption is implemented with an Ideal Cipher on the appropriate message domain, which consists of one-time public keys and/or protocol messages of the underlying otkAKE.

Note that S and C start on resp. inputs A and a and run the following subprotocol: (1) S picks a one-time key pair (b, B) and sends B to C, and (2) the two run otkAKE on resp. (a, B) and (b, A). This subprotocol forms an Authenticated Key Exchange with *unilateral authentication* (ua-KE), where C is authenticated to S but not vice versa.



(a) EKE.v2: sequential, with initiator S

(b) OKAPE-HMQV

Figure 4.1: Symmetric PAKE: *EKE* (a) vs. our asymmetric PAKE's (b)

These parallels are easy to see in Figure 4.1.[2] Since both EKE and our protocols are compilers, resp. from KE and ua-KE, we highlight the underlying uDH instantiation of KE and the one-pass HMQV instantiation of ua-KE in these figures in blue. The choice of variable names g^x and g^b in the Diffie-Hellman key agreement comes from one-pass HMQV, where g^a and g^b are resp. the permanent public key of C and the one-time public key of S, while g^x is the Diffie-Hellman contribution of C. Intuitively, a corresponding g^y contribution of S is not needed because the *ephemeral* key g^b already plays this role.

[2]Actual protocols diverge from Fig. 4.1 in some technicalities, e.g. session key derivation uses a hash of k, but crucially H inputs include a *salt* in OKAPE-HMQV: We come back to this last point below.

The security of our aPAKEs holds for essentially the same reasons as the security of EKE: (1) security against passive attackers holds regardless of pw by the passive security of the underlying (ua-)KE; (2) if encryption is an ideal cipher then any ciphertext sent by an attacker to C decrypts to a random group element $B' = g^{b'}$ on all passwords except the one used by an attacker in encryption, so an attack on such sessions would be an attack on a passively observed otkAKE instance; (3) the attacker can encrypt a chosen g^b value under a single password, but in the IC model the simulator can observe this and extract a unique password guess which the attacker tests in such protocol instance; (4) in OKAPE the client's key confirmation message commits the attacker to a session key, which implies a single input pair (a, B) for which this session key is correct, which in turn commits to a single password from which (a, B) are derived.

Although our protocols can be seen as applications of EKE compiler to ua-KE, we analyze them as compilers from otkAKE for several reasons: First, otkAKE is a simpler notion which can be realized with a single protocol flow; Second, otkAKE yields ua-KE (see above) while the converse is not clear; Third, setting the boundary around otkAKE lets us treat it as a black box in OKAPE compiler, because S's one time key g^b, which is the only part that OKAPE wraps using IC encryption is an input to otkAKE, and not its protocol message.

Similarities to OPAQUE and KHAPE. Our protocols are also closely related to saPAKE protocol OPAQUE [88] and aPAKE protocol KHAPE [74]. Both of these protocols were compilers from AKE (the OPAQUE protocol in addition uses an Oblivious PRF), where passwords are used to encrypt the client's private key a and the server's public key B, the corresponding keys A and b are held in a password file held by S for this client C, and the key establishment comes from AKE run on these inputs. Protocol KHAPE can be seen as a variant of OPAQUE without the Oblivious PRF. In that case security degrades from saPAKE to aPAKE, but the resulting aPAKE can have minimal cost (i.e. $\approx$ KE) if C's AKE inputs (a, B) are delivered from S to C in an envelope, IC-encrypted under the password,

and if the AKE protocol is *key-hiding*, i.e. even an active attacker cannot tell what keys $(K_{\mathsf{P}}, pk_{\mathsf{CP}})$ an attacked party P assumes except if the attacker knows the corresponding pair $(pk_{\mathsf{P}}, K_{\mathsf{CP}})$. The reason the KHAPE compiler needs the key-hiding property of AKE is to avoid off-line attacks, because if each password decrypts the envelope sent to the client into some pair (a', B'), there must be no way to test which pair corresponds to either the client or the server keys unless via an active attack which tests at most one of these choices.

Our compilers OKAPE is a *refinements of the KHAPE compiler*: First, instead of permanent envelope in the password file that encrypts (and authenticates) a permanent server public key g^b, we ask the server to create one-time key per each execution, and IC-encrypt it under a password hash stored in the password file. Replacing key-hiding AKE with key-hiding one-time-key AKE reduces complexity because it can be instantiated with a single C-to-S message. In addition, the IC encryption with subsequent otkAKE together implement implicit S-to-C authentication: If the attacker does not encrypt $B = g^b$ under C's password then C will decrypt it into a random key $B' = g^{b'}$, for which the attacker cannot compute the corresponding session key because it does not know b'. This lets us eliminate the S-to-C key confirmation message in KHAPE and leads to OKAPE.

Note 1: Salted and unsalted password hashes vs. round complexity. The UC aPAKE model of Gentry et al. [72] does not enforce salting of password hashes, which allows their precomputation and an immediate look-up once the server storage is breached. By contrast, Jarecki et al. [88] proposed a UC *strong* aPAKE model (saPAKE), where each password file includes a random and *private* salt value s, and the password hash involves this salt and cannot be precomputed without it. Our protocols OKAPE is just aPAKE, not saPAKE, but they can support *public* salting of the password hash, which has security advantages over unsalted hash. Looking more closely, the aPAKE model of [72] enforces that a single real-world offline dictionary attack test corresponds not only to a single password guess pw^* but also a single tuple $(\mathsf{S}, \mathsf{uid})$ where S is an identifier of a server S and uid is a *userID* with which

S associates a password file. (This can be seen in command $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ to the aPAKE functionality of [72], included in Fig. 2.3 in Section 2.3.3.) This means that a password hash in UC aPAKE, at least as defined by [72], cannot be implemented e.g. simply as $h = \mathsf{H}(pw)$ but in the very least as $h = \mathsf{H}(\mathsf{S}, \mathsf{uid}, pw)$, so that a single H computation corresponds to a single password guess pw and a single account (S, uid). However, such implementation has some negative implications, stemming from the fact that C has to know values $(\mathsf{S}, \mathsf{uid})$ in the protocol. Tying such application-layer values in a cryptographic protocol can be problematic. For example, in some applications it might be fine to equate S with e.g. the server's domain name, but it would be then impossible to modify it, since all users would have to reinitialize and recompute their password hashes. An alternative generic implementation is to use (semi) *public salts* as follows: S can associate each uid account with a random salt s, set the password hash as $h = \mathsf{H}(pw|s)$, attach s in the first S-to-C aPAKE message, and the two parties can then run an *unsalted* aPAKE on a modified password $pw' = pw|s$. Since each s is associated with a unique $(\mathsf{S}, \mathsf{uid})$ pair, each H computation still corresponds to a unique $(\mathsf{S}, \mathsf{uid}, pw)$ tuple, but C does not need values $(\mathsf{S}, \mathsf{uid})$ within the aPAKE protocol, and password hashes do not have to change with changes to identifiers S or uid. Moreover, if the aPAKE protocol runs over a TLS connection then an adversary can find s only via an online interaction with S, and it needs to know the user ID string uid for S to retrieve the uid-indexed password file and send s out. Even better, if clients update the (s, h) values at each login, then value s the adversary compromises for some user will be obsolete after that user authenticates to S.

However, this implementation requires interaction. Since S sends the first message in OKAPE, attaching s to S's message does not influence the round complexity of OKAPE, and this is indeed how we implement password hashes in that protocol, see Section 6.2. Every unsalted aPAKE can be transformed to *publicly salted* in this way, but for many aPAKEs, this would imply additional communication rounds.

Note 2: Implicit and explicit authentication vs. round complexity. explicit entity authentication requires each party computes a key and the security implies that only a party with proper credentials can compute that key as well, but they do not get a confirmation that their counterparty can compute the same key and thus is indeed the party they meant to establish a connection with. Key confirmation can be added to any KE protocol, but it adds a round of communication. Our three-round (if C initializes) aPAKE protocol OKAPE has only C-to-S entity authentication, and adding S-to-C entity authentication would make it a four-round protocol. Therefore the round-reduction advantage of OKAPE over protocol KHAPE of [74] will benefit only those applications where C can use the session key without waiting for S's key confirmation message.

Note 3: Current costs of ideal cipher on groups. Just like EKE [28, 26], our protocols rely on an ideal cipher on group elemnets. Implementing an ideal cipher on elliptic curve groups, which are of most interest for current aPAKE proposals, is non-trivial and current techniques for implementing them incur non-negligible costs in computation and sometimes in bandwidth expansion as well. We discuss several implementation options for group IC in Section 3.8, but to give an example, using the Elligator2 method [32] each IC operation can cost ≈10-15% of 1vb exp and it requires resampling of the encrypted random group element with probability 1/2. Thus we can estimate the total computational cost of $\mathsf{OKAPE\text{-}HMQV}$ with this IC implementation as (expected) 2fb+1.15vb for S and 1fb+1.15vb for C. However, the overhead of IC might be significantly smaller in the case of other settings of interest, like lattice cryptosystems.

4.2 Key-hiding one-time-key AKE

We define key-hiding *one-time-key* Authenticated Key Exchange (otkAKE), as an asymmetric variant of the universally composable key-hiding AKE defined in [74]. We denote

the otkAKE functionality $\mathcal{F}_{\mathsf{otkAKE}}$ and we include it in Figure 4.2. An AKE functionality allows parties to generate public key pairs (this is modeled by environment query Init to the functionality). These keys can be compromised, modeled by adversarial query Compromise. However, this is the key difference between our (key-hiding) otkAKE functionality and the (key-hiding) AKE functionality of [74], here we distinguish two types of keys, the long-term keys which can be compromised by the adversary, and the ephemeral keys which cannot. We arbitrarily call the first type "client keys" and the second "server keys" because this is how we will use an otkAKE protocol in the context of our otkAKE-to-aPAKE compiler in Section 6.2, i.e. clients will use long-term keys and servers will use ephemeral keys in both of these applications of otkAKE.

As in [74], any party P holding a key pair indexed by the public key pk_{P} , whether a long-term one or an ephemeral one, can start a session using such key, and using also some key pk_{CP} as the public key of the counterparty that P expends on this session. This is modeled by the environment's command $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P, where sid is the unique session identifier, CP is the supposed identifier of the counterparty, and role is either 1 or 2, defining if P is supposed to run the long-term-key party or the ephemeral-key party. (As we can see below, the protocols realizing this functionality can be asymmetric, so parties act differently based on that role bit.) As in [74], the functionality marks this session as initially fresh, creates an appropriate session record and picks a random function $R_{\mathsf{P}}^{\mathsf{sid}}$ (whose meaning we will explain shortly). Crucially the functionality only sends $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role})$ to the adversary, i.e. the adversary only learns which party P wants to authenticate, which party CP they intend to communicate with, what session identifier sid they use, and whether they play the client and the server role, but the adversary does *not* learn the *keys* this party uses, neither their own key pk_{P} nor the key pk_{CP} this party expects of its counterparty. This, exactly as in [74], models the *key-hiding* property of the AKE's which are required in our AKE-to-aPAKE compiler constructions.

PK stores all public keys created in Init; CPK stores all compromised keys;
PK^1_{P} stores P's permanent public keys; PK^2_{P} stores P's ephemeral public keys;

Keys: Initialization and Attacks

On $(\mathsf{Init}, \mathsf{role})$ from P:

If $\mathsf{role} \in \{1, 2\}$ send $(\mathsf{Init}, \mathsf{P}, \mathsf{role})$ to $\mathcal{A}$, let $\mathcal{A}$ specify pk s.t. $pk \notin PK$, add pk to PK and $PK^{\mathsf{role}}_{\mathsf{P}}$, and output (Init, pk) to P. If P is corrupt then add pk to CPK.

On $(\mathsf{Compromise}, \mathsf{P}, pk)$ from $\mathcal{A}$: [*this query must be approved by the environment*]
If $pk \in PK^1_{\mathsf{P}}$ then add pk to CPK.

Login Sessions: Initialization and Attacks

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ from P:

If $pk_{\mathsf{P}} \in PK^{\mathsf{role}}_{\mathsf{P}}$ and there is no prior session record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \cdot \rangle$ then:

- create session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp \rangle$ marked fresh;
- if $\mathsf{role} = 1$ and $pk_{\mathsf{CP}} \notin PK^2_{\mathsf{CP}}$ then re-label this record as $\mathsf{interfered}$;
- initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : \{0,1\}^3 \to \{0,1\}^\kappa$;
- send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role})$ to $\mathcal{A}$.

On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$:

If there is session $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \perp \rangle$ marked fresh then change it to $\mathsf{interfered}$.

Login Sessions: Key Establishment

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ from $\mathcal{A}$:

If $\exists$ session record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp \rangle$ then:

- if rec is marked fresh: If $\exists$ record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk_{\mathsf{CP}}, pk_{\mathsf{P}}, \mathsf{role}', k' \rangle$ marked fresh s.t. $\mathsf{role}' \neq \mathsf{role}$ and $k' \neq \perp$ then set $k \leftarrow k'$, else pick $k \xleftarrow{r} \{0,1\}^\kappa$;
- if rec is marked $\mathsf{interfered}$ then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$;
- update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P.

Session-Key Query

On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$ from $\mathcal{A}$:

If $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \cdot \rangle$ and $pk' \notin (PK \setminus CPK)$ then send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$.

Figure 4.2: $\mathcal{F}_{\mathsf{otkAKE}}$: Functionality for key-hiding one-time key AKE

Next, if an adversary actively attacks session $\mathsf{P}^{\mathsf{sid}}$, as opposed to passively observing its interaction with some other session $\mathsf{CP}^{\mathsf{sid}}$, this is modeled by the adversarial query $\mathsf{Interfere}$, and its effect is that session $\mathsf{P}^{\mathsf{sid}}$ is marked as $\mathsf{interfered}$. The consequence of this marking comes in when the session terminates (i.e. if the adversary delivers all messages this party expects) and outputs a key, which is modeled by adversarial query NewKey. Namely, if a session is fresh, i.e. it was not actively attacked, then the functionality picks its output session key k as a random string. In other words, this key is secure because there is no interface which allows the adversary to get any information about it. If the adversary passively connects two sessions, e.g. $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$, by honestly exchanging their messages, then $\mathcal{F}_{\mathsf{otkAKE}}$ will notice at the NewKey processing that there are two sessions $(\mathsf{P}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role})$ $(\mathsf{CP}, \mathsf{sid}, \mathsf{P}, pk'_{\mathsf{P}}, pk'_{\mathsf{CP}}, \mathsf{role}')$ that run on matching keys, i.e. $pk_{\mathsf{CP}} = pk'_{\mathsf{P}}$ and $pk'_{\mathsf{CP}} = pk_{\mathsf{P}}$, and complementary roles, i.e. $\mathsf{role} \neq \mathsf{role}'$, then $\mathcal{F}_{\mathsf{otkAKE}}$ sets the key of the session that terminates last as a copy of the one that terminated first. This is indeed as it should be: If two parties run AKE on matching inputs and keys and their messages are delivered without interference they should output the same key.

However, if session $\mathsf{P}^{\mathsf{sid}}$ has been actively attacked, hence it is marked $\mathsf{interfered}$, the session key k output by $\mathsf{P}^{\mathsf{sid}}$ is determined by the random function $R^{\mathsf{sid}}_{\mathsf{P}}$. Specifically, the key will be assigned as the value of $R^{\mathsf{sid}}_{\mathsf{P}}$ on a tuple of three inputs: (1) P's own key pk_{P}, (2) the counterparty's key pk_{CP} which P assumes, and (3) the protocol transcript α which w.l.o.g. is determined by the adversary on this session. This is a non-standard way of modeling KE functionalities, but it suffices for our applications and it allows for inexpensive and communication-minimal implementations as we exhibit with protocols 2DH and one-pass HMQV below. The intuition is that this assures that for any protocol transcript the adversary chooses, each key pair $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ which P can use corresponds to an independent session key output of P. Some of these keys can be computed by the adversary via interface $\mathsf{ComputeKey}$: The adversary can use it to compute the key P would output on a given transcript α and a given pair $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk, pk')$ but only if pk' is either compromised or it is an adversarial

key, hence w.l.o.g. we assume the adversary knows the corresponding secret key.

Here is also where our key-hiding one-time-key AKE diverges from the key-hiding AKE notion of [74]: If session $\mathsf{P}^{\mathsf{sid}}$ runs with a client-role then its session key output is guaranteed secure if their assumed counterparty's key pk_{CP} is indeed an ephemeral key of the intended counterparty. Since such keys cannot be compromised, a $\mathsf{ComputeKey}$ query with $pk' = pk_{\mathsf{CP}}$ will fail the criterion that pk' is compromised or adversarial, hence the adversary has *no interface to learn* P*'s output session key.* However, if the environment (i.e. the higher-level application, like either of our compilers, which utilizes the otkAKE subprotocol) asks $\mathsf{P}^{\mathsf{sid}}$ to run on pk_{CP} which is *not* an ephemeral key of the intended counterparty then $\mathcal{F}_{\mathsf{otkAKE}}$ treats such session as automatically attacked, and marks it $\mathsf{interfered}$. Such session's output key will be computed as $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$, and whether or not the adversary can recompute this key via the $\mathsf{ComputeKey}$ interface depends on whether this (potentially non-ephemeral) key pk_{CP} is compromised or adversarial.

The security of our otkAKE protocols, 2DH and one-pass HMQV, are based on hardness of Gap CDH problem. Recall that Gap CDH is defined as follows: Let g generates a cyclic group $\mathbb{G}$ of prime order p. The Computational Diffie-Hellman (CDH) assumption on $\mathbb{G}$ states that given $(X, Y) = (g^x, g^y)$ for $(x, y) \overset{r}{\leftarrow} (\mathbb{Z}_p)^2$ it's hard to find $\mathsf{cdh}_g(X, Y) = g^{xy}$. The Gap CDH assumption states that CDH is hard even if adversary has access to a Decisional Diffie-Hellman oracle ddh_g, which on input (A, B, C) returns 1 if $C = \mathsf{cdh}_g(A, B)$ and 0 otherwise.

4.2.1 2DH as key-hiding one-time-key AKE

We show that key-hiding one-time-key AKE can be instantiated with a "one-pass" variant of the 3DH AKE protocol. 3DH is an implicitly authenticated key exchange used as the basis of the X3DH protocol [108] that underlies the Signal encrypted communication application.

3DH consists of a plain Diffie-Hellman exchange which is authenticated by combining the ephemeral and long-term key of both peers. Specifically, if (a, A) and (b, B) are the long-term key pairs of two communicating parties C and S, and (x, X) and (y, Y) are their ephemeral DH values, then 3DH computes the session key as a hash of the *triple* of Diffie-Hellman values, (g^{xb}, g^{ay}, g^{xy}).

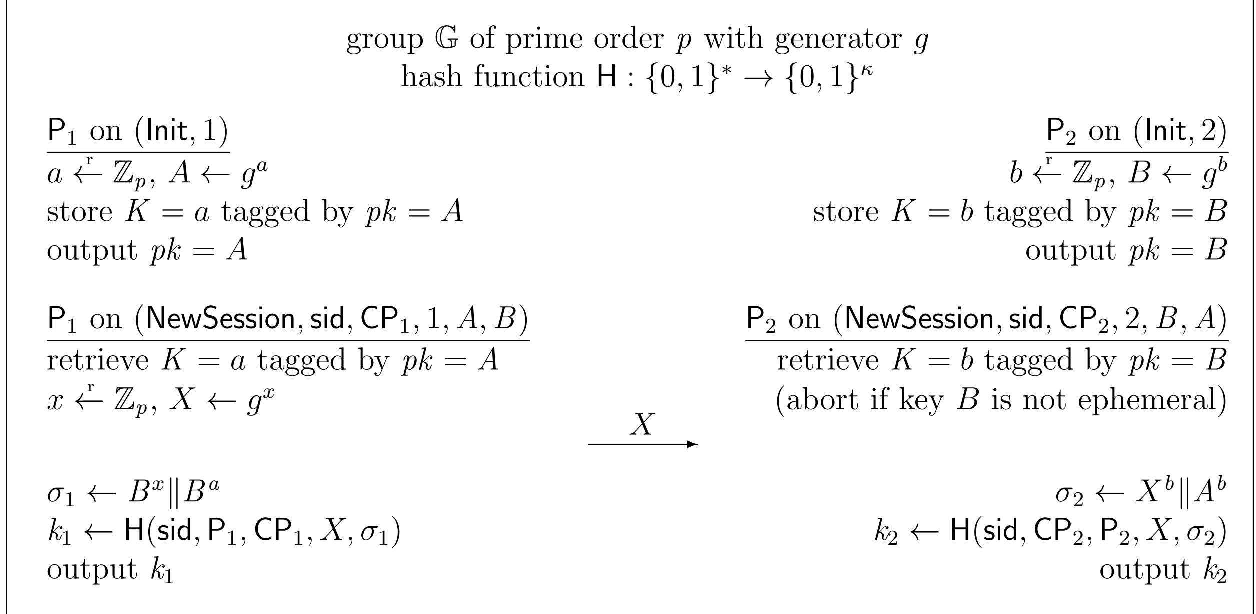


Figure 4.3: otkAKE protocol $2DH$

This protocol was shown to realize the key-hiding AKE functionality in [74], and here we show that a one-pass version of this protocol, which we call *2DH*, realizes the key-hiding one-time-key AKE functionality $\mathcal{F}_{\mathsf{otkAKE}}$ defined above. In this modified setting key (b, B) is a one-time key of party S, and hence it can play a double-role as S authenticator *and* its ephemeral DH contribution. Therefore the only additional ephemeral key needed is the (x, X) value provided by C, and 2DH will compute the session key as a (hash of) the *pair* of DH values, (g^{xb}, g^{ab}). See Figure 4.3 were we describe the 2DH protocol in more detail. In that figure we assume that both C's key (a, A) and S's key (b, B) were created prior to protocol execution, but we note that S's key must be a one-time, i.e. ephemeral, key, so in practice it should be created just before the protocol starts and erased once the protocol executes.

We capture the security property of 2DH in the following theorem:

Theorem 4.1. *Protocol 2DH shown in Figure 4.3 realizes functionality $\mathcal{F}_{\mathsf{otkAKE}}$, assuming that the Gap CDH assumption holds on group $\mathbb{G}$ and H is a random oracle.*

The proof of the above theorem is a close variant of the proof given in [74] that 3DH realizes the key-hiding AKE functionality (where both parties use permanent keys).

Below we show the full proof for theorem 4.1. Standardly we assume that real-world adversary $\mathcal{A}$ is a subroutine of the environment $\mathcal{Z}$, therefore the sole party that interacts with Games 0 or 7 is $\mathcal{Z}$, who issue commands Init and $\mathsf{NewSession}$ to honest parties P, adaptively compromise public keys, and use $\mathcal{A}$ to send protocol messages Z to honest server sessions and make hash function H queries. The proof follows a standard strategy by showing a sequence of games that bridge between Game 0 and Game 7, where at each transition we argue that the change is indistinguishable to $\mathcal{Z}$. We use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if we show that $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G7}]|$ is negligible under the stated assumptions.

Notion. To make the real-world interaction in Figure 4.4 more concise, we adopt a notation where we use variable $W = g^w$ to denote the message which party C sends out, and variable Z to denote the message S receives.

Throughout the proof we use $\mathsf{P}^{\mathsf{sid}}$ to denote a session of party P with identifier sid. We use $v_{\mathsf{P}}^{\mathsf{sid}}$ to denote a local variable v pertaining to session $\mathsf{P}^{\mathsf{sid}}$ or a message v which this session receives, and whenever identifier sid is clear from the context we write v_{P} instead of $v_{\mathsf{P}}^{\mathsf{sid}}$. Note that session $\mathsf{CP}^{\mathsf{sid}}$ is uniquely defined for every session $\mathsf{P}^{\mathsf{sid}}$ by setting $\mathsf{CP} = \mathsf{CP}_{\mathsf{P}}^{\mathsf{sid}}$, and we will implicitly assume in the proof that a counterparty's session is defined in this way.

For a fixed environment $\mathcal{Z}$, let q_{K} and q_{ses} be (the upper-bounds on) the number of resp. keys and sessions initialized by $\mathcal{Z}$, let q_{H} be the number of H oracle queries $\mathcal{Z}$ makes, and

let $\epsilon^{\mathcal{Z}}_{\text{g-cdh}}$ be the maximum advantage in solving Gap CDH in $\mathbb{G}$ of an algorithm that makes q_{H} DDH oracle queries and uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{H}} + q_{\text{ses}})$ exponentiations in $\mathbb{G}$.

Initialization: Initialize empty lists $PK^1_{\mathsf{P}}, PK^2_{\mathsf{P}}, KL_{\mathsf{P}}$ for each P

On message $(\mathsf{Init}, \mathsf{role})$ to P:
If $\mathsf{role} \in \{1, 2\}$ then pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add pk to $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$:
 if $\mathsf{role} = 1$ and $pk_{\mathsf{P}} \in PK^1_{\mathsf{P}}$, pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K, pk_{\mathsf{CP}}, w \rangle$, output $W = g^w$, set $\sigma \leftarrow ({pk_{\mathsf{CP}}}^w \| {pk_{\mathsf{CP}}}^K)$, $k \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, W, \sigma)$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$
 else if $\mathsf{role} = 2$ and $pk_{\mathsf{P}} \in PK^2_{\mathsf{P}}$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K, pk_{\mathsf{CP}}, \perp \rangle$

On message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A, \perp \rangle$, set $\sigma \leftarrow (Z^b \| A^b)$, $k \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \sigma)$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 4.4: 2DH: Environment's view of real-world interaction (Game 0)

Proof. The 2DH proof below shows the indistinguishability between the real-world game (Game 0) shown in Figure 4.4, which captures an interaction with parties running the 2DH protocol, and the ideal-world game (Game 7) shown in Figure 4.6, which is defined by a composition of SIM and functionality $\mathcal{F}_{\mathsf{otkAKE}}$. For each AKE session we define function $R^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', \perp)$ which is used by session $\mathsf{C}^{\mathsf{sid}}$(resp. $R^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', Z)$ used by $\mathsf{S}^{\mathsf{sid}}$, Z is message S receives) to compute its session key. Below we define function $\mathsf{2DH}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ for session $\mathsf{P}^{\mathsf{sid}}$ running on inputs $(\mathsf{sid}, \mathsf{CP}, pk, pk', \alpha)$, i.e. pk is its own public key, pk' is the public key of its intended counterparty, and since it's asymmetric, α can be either $\perp$ if $\mathsf{P} = \mathsf{C}$ or Z if $\mathsf{P} = \mathsf{S}$:

Initialization: Initialize an empty list KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P}, \mathsf{role})$ from $\mathcal{F}$:
pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (K, pk) \in KL_{\mathsf{P}}$ send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$ and send K to $\mathcal{A}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role})$ from $\mathcal{F}$:
if $\mathsf{role} = 1$: pick $w \xleftarrow{r} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w \rangle$, send $W = g^w$ to $\mathcal{A}$
send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}$
else store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, \perp \rangle$

On $\mathcal{A}$'s message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, \perp \rangle$:
if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, z \rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$ to $\mathcal{F}$
send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, Z)$ to $\mathcal{F}$

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ to random oracle H:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t. $(X, \sigma) = (g^x, (B^x \| B^a))$ for some B,
send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, \perp)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, \perp \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t. $\sigma = (X^b \| A^b)$ for some A, send
$(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 4.5: Simulator SIM showing that 2DH realizes $\mathcal{F}_{\mathsf{otkAKE}}$ (abbreviated "$\mathcal{F}$")

$$\mathsf{2DH}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp) = \mathsf{cdh}_g(W_{\mathsf{C}}^{\mathsf{sid}}, pk') \| \mathsf{cdh}_g(pk, pk') \tag{4.1}$$

$$\mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z) = \mathsf{cdh}_g(Z, pk) \| \mathsf{cdh}_g(pk, pk') \tag{4.2}$$

$$R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp) = \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, W_{\mathsf{C}}^{\mathsf{sid}}, \mathsf{2DH}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp)) \tag{4.3}$$

$$R_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z) = \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)) \tag{4.4}$$

GAME 0 *(real world)*: The real-world game is the real world view of executing protocol 4.3.

GAME 1 *(past* H *queries are irrelevant to new sessions)*: Game 1 adds an abort if $\mathsf{NewSession}$ initializes session $\mathsf{C}^{\mathsf{sid}}$ with $W = g^w$ s.t. H has been queried on any tuple of the form $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, W, \cdot)$. Since each H query can only pertain to $\mathsf{C}^{\mathsf{sid}}$, there are at most q_{H} such queries, and $w \xleftarrow{r} \mathbb{Z}_p$, we have:

$$|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq q_{\mathsf{H}}/p$$

GAME 2 *(programming* $R^{\mathsf{sid}}_{\mathsf{P}}$ *values into* H *outputs)*: Define sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ to be *matching* if $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}} = \mathsf{S}$ and $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}} = \mathsf{C}$. Note that for any matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ and any public keys A, B, correctness of 2DH implies that $R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X_{\mathsf{C}})$. While in equation (4.3)(4.4) we defined function $R^{\mathsf{sid}}_{\mathsf{P}}$ in terms of hash H, in Game 2 we set H outputs using appropriately chosen functions $R^{\mathsf{sid}}_{\mathsf{P}}$. For every pair of matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ of role $1, 2$ consider a pair of random functions $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}} : (\mathbb{G})^3 \to \{0,1\}^\kappa$ s.t.

$$R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X^{\mathsf{sid}}_{\mathsf{C}}) \quad \text{for all } A, B \in \mathbb{G} \tag{4.5}$$

More precisely, for any session $\mathsf{P}^{\mathsf{sid}}$ with no matching session, $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function, and for $\mathsf{P}^{\mathsf{sid}}$ for which a prior matching session exists $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function subjects to constraint (4.5). Let PK be the list of all public keys generated so far, and PK_{P} be the set of keys generated for P. Let $PK^+(\mathsf{P}^{\mathsf{sid}})$ stand for $PK \cup \{pk_{\mathsf{CP}}\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$. (If $pk_{\mathsf{CP}} \in PK$ then $PK^+(\mathsf{P}^{\mathsf{sid}}) = PK$.) Consider an oracle H which responds to each new query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ as follows:

1. If $\exists\ \mathsf{C}^{\mathsf{sid}}$ s.t. $(\mathsf{S}, X) = (\mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}}, X^{\mathsf{sid}}_{\mathsf{C}})$, and $\exists\ A, B$ s.t. $A \in PK_{\mathsf{C}}$, $B \in PK^+(\mathsf{C}^{\mathsf{sid}})$, and $\mathsf{2DH}^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp) = \sigma$, then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp)$
2. If $\exists\ \mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{C} = \mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}}$, and $\exists\ B, A$ s.t. $B \in PK_{\mathsf{S}}$, $A \in PK^+(\mathsf{S}^{\mathsf{sid}})$, and X satisfies $\mathsf{2DH}^{\mathsf{sid}}_{\mathsf{S}}(B, A, X) = \sigma$, then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$

Initialization: Initialize empty lists: PK, PK^1_{P}, PK^2_{P}, CPK, and KL_{P} for all P

On message $(\mathsf{Init}, \mathsf{role})$ to P:

set $K \xleftarrow{r} \mathbb{Z}_p$, $pk \leftarrow g^K$, send (Init, pk) to P, add pk to PK and $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:

If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then add pk to CPK and output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:

if $\exists\ (K, pk_{\mathsf{P}}) \in KL_P$ then:
 initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$
 if $\mathsf{role} = 1$, pick $w \xleftarrow{r} \mathbb{Z}_p$, $k \xleftarrow{r} \{0,1\}^\kappa$,
 write rec $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k\rangle$ as fresh, output $W = g^w$,
 if $pk_{\mathsf{CP}} \notin PK^2_{\mathsf{CP}}$ then mark rec $\mathsf{interfered}$ and set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp)$
 send $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P
 else write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp, \perp\rangle$ as fresh

On message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message is processed):

if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, B, A, 2, \perp, \perp\rangle$:
 if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, A', B', 1, z, k'\rangle$ s.t. $g^z = Z$
  then if rec' is fresh, $(B, A) = (B', A')$, and $k' \neq \perp$:
   then $k \leftarrow k'$
   else $k \xleftarrow{r} \{0,1\}^\kappa$
  else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, Z)$ and re-label rec as $\mathsf{interfered}$
 update rec to $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, B, A, 2, \perp, k\rangle$, send $(\mathsf{NewKey}, \mathsf{sid}, k)$ to S

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$:

if $\exists\ \langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot\rangle$ s.t. $(X, \sigma) = (g^x, (B^x \| B^a))$ for some $(a, A) \in KL_{\mathsf{C}}$ and B s.t. $B \in CPK$ or $B \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp)$
2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, \perp, \cdot\rangle$ s.t. $\sigma = (X^b \| A^b)$ for some $(b, B) \in KL_{\mathsf{S}}$ and A s.t. $A \in CPK$ or $A \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$

add $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ to $\mathsf{T_H}$ and output k

Figure 4.6: 2DH: Environment's view of ideal-world interaction

3. In any other case sample $k \xleftarrow{r} \{0,1\}^\kappa$

Since the game knows each key pair $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ generated for each P, and the ephemeral state w of each session $\mathsf{C}^{\mathsf{sid}}$, it can decide for any Z, pk' if $\sigma = \mathsf{2DH}^{\mathsf{sid}}_{\mathsf{C}}(A, pk', \bot) = (pk')^w \| pk'^a$ and if $\sigma = \mathsf{2DH}^{\mathsf{sid}}_{\mathsf{S}}(B, pk', Z) = Z^b \| pk'^b$. Note that each value of $R^{\mathsf{sid}}_{\mathsf{P}}$ is used to program H on at most one query. Also if the same hash query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ matches both the client-side equation and the server-side equation, i.e. if

$$\sigma = \mathsf{2DH}^{\mathsf{sid}}_{\mathsf{C}}(A, B', \bot) = g^{b'x} \| g^{b'a} = g^{bx} \| g^{ba'} = \mathsf{2DH}^{\mathsf{sid}}_{\mathsf{S}}(B, A', X)$$

where $A \in PK_{\mathsf{C}}, B' \in PK^+(\mathsf{C}^{\mathsf{sid}}), B \in PK_{\mathsf{S}}, A' \in PK^+(\mathsf{S}^{\mathsf{sid}})$, then it implies that both parties must use correct counterparty keys, i.e. that $(A', B') = (A, B)$, which guarantees $R^{\mathsf{sid}}_{\mathsf{C}}$ and $R^{\mathsf{sid}}_{\mathsf{S}}$ programs H to the same value in matching sessions. Thus it follows that:

$$|\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]|$$

GAME 3 *(direct programming of session keys using random functions $R^{\mathsf{sid}}_{\mathsf{P}}$)*: In Game 3 we make the following changes: (1) We mark each initialized client session $\mathsf{C}^{\mathsf{sid}}$ with intended honestly generated ephemeral (resp. adversarially generated or permanent) key as fresh (resp. $\mathsf{interfered}$). We mark each $\mathsf{S}^{\mathsf{sid}}$ similarly, and also re-label it as $\mathsf{interfered}$ if the message Z this server session receives does not equal to the message sent by the matching session $\mathsf{C}^{\mathsf{sid}}$, i.e. if $Z^{\mathsf{sid}}_{\mathsf{S}} \neq W^{\mathsf{sid}}_{\mathsf{C}}$. (2) if session $\mathsf{P}^{\mathsf{sid}}$ runs on its own key pair $(K_{\mathsf{P}}, pk_{\mathsf{P}})$ and intended counterparty public key pk_{CP}, we say that it runs "under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$". Using this book-keeping, Game 3 modifies session-key computation for session $\mathsf{P}^{\mathsf{sid}}$ which runs under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ as follows:

1. If $k^{\mathsf{sid}}_{\mathsf{CP}} \neq \bot$, sessions $\mathsf{P}^{\mathsf{sid}}, \mathsf{CP}^{\mathsf{sid}}$ are fresh and *matching*, and $\mathsf{CP}^{\mathsf{sid}}$ runs under keys $(pk_{\mathsf{CP}}, pk_{\mathsf{P}})$, then $k^{\mathsf{sid}}_{\mathsf{P}} \leftarrow k^{\mathsf{sid}}_{\mathsf{CP}}$

2. In any other case, set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, \perp)$ if $\mathsf{P}^{\mathsf{sid}}$ is playing the role of a client, otherwise set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$.

We argue that this change makes no difference to the environment. Take server side as example, in Game 2 the session key $k_{\mathsf{S}}^{\mathsf{sid}}$ is computed as $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \sigma)$ for $\sigma = \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$. However, H on such input is programmed in Game 2 to output $R_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$ if $\sigma = \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$ for any $A \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$. Since A used by $\mathsf{S}^{\mathsf{sid}}$ is by definition in set $PK^{+}(\mathsf{S}^{\mathsf{sid}})$, setting $k_{\mathsf{S}}^{\mathsf{sid}}$ directly as $R_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$ only short-circuits this process. The client side is symmetric. Finally, since $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ are correlated, setting $k_{\mathsf{C}}^{\mathsf{sid}}$ as $k_{\mathsf{S}}^{\mathsf{sid}}$ or vice versa, in the case both are fresh and *matching*, also does not change the game. Thus we conclude:

$$|\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]|$$

GAME 4 *(abort on* H *queries for passive sessions)*: Define a passive session as a session that is fresh. Equivalently, these are the client sessions that receive honestly generated ephemeral counterparty public keys and the server sessions that run on permanent client keys and receive an unmodified client message. We add an abort on adversarial H queries that trigger key computations for passive sessions, i.e. if the environment queries H triggering evaluation of (1) $R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp)$ or (2) $R_{\mathsf{S}}^{\mathsf{sid}}(pk', pk, Z)$ for any $pk \in PK_{\mathsf{C}}^{1}$, any $pk' \in PK_{\mathsf{S}}^{2}$ and $Z = W_{\mathsf{C}}^{\mathsf{sid}}$ where $\mathsf{C}^{\mathsf{sid}}$ is the matching session of $\mathsf{S}^{\mathsf{sid}}$. Our goal is to avoid allowing the adversary to query output keys to passive sessions. By the code of oracle H in Game 2 the call to $R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp)$ is triggered only if client side H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, W, \sigma)$ satisfies $\sigma = \mathsf{cdh}_g(W, pk') \| \mathsf{cdh}_g(pk, pk')$, and symmetrically $R_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$ is triggered only if H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \sigma)$ satisfies $\sigma = \mathsf{cdh}_g(Z, pk) \| \mathsf{cdh}_g(pk, pk')$.

We define Bad as the event where such an H query happens and we show that if it does then we can solve Gap CDH. On input a CDH challenge $(\bar{X}, \bar{B})$, the reduction $\mathcal{R}$ sets each $X_{\mathsf{C}}^{\mathsf{sid}}$ as $\bar{X}^{s}$ for random s. $\mathcal{R}$ also picks all keys (a, A) as in Game 0, and sets each server public

key $pk = B = \bar{B}^t$ for random t. Since keys pk are one-time and uncompromisable, $\mathcal{R}$ can answer any compromise key request made by the adversary by returning the corresponding a. Also, although $\mathcal{R}$ doesn't know $x = s \cdot \bar{x}$ and $b = t \cdot \bar{b}$ corresponding to message X and public keys B, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{b} = \mathsf{dlog}_g(\bar{B})$, it can use the DDH oracle to emulate the way Game 3 services every H queries (not only those for passive sessions): to test if client side H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ for $X = \bar{X}^s$ satisfies $\sigma = (L \| M) = (B^x \| B^a)$ for $x = s \cdot \bar{x}$, any private key a, and some $B \in PK^+(\mathsf{C}^{\mathsf{sid}})$, $\mathcal{R}$ checks if $L = \mathsf{cdh}_g(X, B)$ and $M = B^a$. Symmetrically, $\mathcal{R}$ tests if server side H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ satisfies $\sigma = (L \| M) = (X^b \| A^b)$ for $A \in PK^+(\mathsf{S}^{\mathsf{sid}})$ and server private key b by checking if $L = \mathsf{cdh}_g(X, B)$ for a public server key $B = \bar{B}^t$ and $M = \mathsf{cdh}_g(A, B)$. Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3. By the above, $\mathcal{R}$ can detect event Bad and then output $L^{1/st} = \mathsf{cdh}_g(\bar{X}, \bar{B})$ as the answer $\mathsf{cdh}_g(\bar{X}, \bar{B})$ to its CDH challenge. It follows that the reduction solves the Gap-CDH problem with probability at least as big as $\Pr[\mathsf{Bad}]$, hence:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Game 5 *(random keys on passively observed sessions)*: We modify the game so that if session $\mathsf{C}^{\mathsf{sid}}$ is initialized as fresh, i.e. if the counterparty key is not adversarial, then it sets $k_{\mathsf{C}}^{\mathsf{sid}} \leftarrow \{0,1\}^\kappa$. Additionally, if the above happens and $\mathsf{S}^{\mathsf{sid}}$ remains fresh by the time $\mathcal{A}$ sends Z to $\mathsf{S}^{\mathsf{sid}}$, then instead of setting $k_{\mathsf{S}}^{\mathsf{sid}} \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$ as in Game 3, we now set $k_{\mathsf{S}}^{\mathsf{sid}} \leftarrow k_{\mathsf{C}}^{\mathsf{sid}}$. Since session $\mathsf{S}^{\mathsf{sid}}$ can remain fresh only if Z it receives was sent by its matching session, i.e. $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$, and by Game 4 oracle H never queries $R_{\mathsf{S}}^{\mathsf{sid}}(B, A, Z)$ for such Z, it follows by randomness of $R_{\mathsf{S}}^{\mathsf{sid}}$ that the modified game remains externally identical.

$$\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$$

GAME 6 *(decorrelating function pairs $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$)*: Let Game 6 be as Game 5, except that functions $R_{\mathsf{S}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$ are chosen without the constraint imposed by equation (4.5). Since by Game 5 neither function is queried on the points which create the correlation imposed by equation (4.5), it follows that:

$$\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$$

GAME 7 *(hash computation consistent only for compromised keys)*: Recall that in Game 6, as in Game 2, server side $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \sigma)$ is defined as $R_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$ if $\sigma = \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$ for some $pk \in PK_{\mathsf{S}}$ and $pk' \in PK^{+}(\mathsf{C}^{\mathsf{sid}})$. In Game 7 we add a condition that this programming of H can occur only if (1) pk' is an *adversarial* key, i.e., it has not been generated by $(\mathsf{Init}, 1)$ or (2) pk' is an honestly generated permanent key, but it has been $\mathsf{compromised}$. These are the two cases in which the adversary can know the secret key corresponding to pk', and we show that these are the only cases when the adversary can compute σ s.t. $\sigma = \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$, and hence trigger the programming of H.

Let CPK be the list of generated public keys who were compromised so far, and let CPK^{+} $(\mathsf{P}^{\mathsf{sid}})$ stand for CPK if the counterparty public key pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ is an honestly generated key, and for $CPK \cup \{pk_{\mathsf{CP}}\}$ if pk_{CP} is adversarially-generated. The modification of Game 7 is that on server side H output is programmed to $R_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$ for pk' s.t. $\sigma = \mathsf{2DH}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$, only if $pk' \in CPK^{+}(\mathsf{S}^{\mathsf{sid}})$. In Game 6, as in Game 2, this programming was done whenever $pk' \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$. Therefore the two games diverge in the case of event Bad_2 defined as server side H query as above for $pk' \in PK \setminus CPK$, i.e. honestly generated and *not* compromised key. We show a reduction $\mathcal{R}$ that solves Gap CDH if Bad_2 occurs. (The corresponding event for the client side was already handled in Game 4 since it corresponds to a passively-observed client session running on an honest one-time server key.)

Bad_2 corresponds to H query on string $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$ for $\sigma = \mathsf{cdh}_g(X, B) \| \mathsf{cdh}_g(A, B)$ where

X is arbitrary, A is some (non compromised) client public key, and B is a one-time and uncompromisable server public key. On input a CDH challenge $(\bar{A}, \bar{B})$, $\mathcal{R}$ picks each protocol message $X = g^x$ for random x of its choice, and sets each $B \leftarrow \bar{B}^t$ for random t. $\mathcal{R}$ also picks all client keys (a, A) as in Game 0, except for the i-th C key $pk[i]$, for a random index $i \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ sets the key generated in the i-th call to $(\mathsf{Init}, 1)$ as $pk[i] \leftarrow \bar{A}$. Let $\mathsf{Bad}_2[i]$ denote event Bad_2 occuring for A which is this i-th key, i.e. $A = \bar{A}$.

As long as key $pk[i]$ is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys (remember that only client keys are compromisable), and it can service H queries as follows: to test client-side σ's, i.e. if $\sigma = (L \| M) = (B^x \| B^a)$, reduction $\mathcal{R}$ tests as in Game 6, except for a that corresponds to public key $\bar{A}$, in which case it tests if $M = \mathsf{cdh}_g(B, \bar{A})$. To test server-side σ's, i.e. if $\sigma = (L \| M) = (X^b \| pk^b)$ for $b = t \cdot \bar{b}$ where $\bar{b} = \mathsf{dlog}_g(\bar{B})$ and $pk \in PK^+(\mathsf{S}^{\mathsf{sid}})$, including the case when $pk = \bar{A}$ or pk is an adversarial key, reduction $\mathcal{R}$ tests if $L = \mathsf{cdh}_g(X, B)$ and $M = \mathsf{cdh}_g(pk, B)$.

Note that $\mathsf{Bad}_2[i]$ can happen only before key $pk[i]$ is compromised, so event $\mathsf{Bad}_2[i]$ occurs in the reduction with the same probability as in Game 6. (If $\mathcal{A}$ asks to compromise $pk[i]$ then $\mathcal{R}$ aborts.) $\mathcal{R}$ can detect event $\mathsf{Bad}_2[i]$ because it occurs if H query involves the public key $pk[i] = \bar{A}$ and σ satisfies the server-side equation for this key, in which case $\mathcal{R}$ can output $M^{1/t} = \mathsf{cdh}_g(\bar{A}, \bar{B})$ thus solving Gap-CDH. If $\mathcal{R}$ picks index i at random it follows that $\Pr[\mathsf{Bad}_2] \leq q_{\mathrm{K}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$. Thus we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq q_{\mathrm{K}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Observe that Game 7 is identical to the ideal-world game shown in Figure 4.4: By Game 6 all functions $R^{\mathsf{sid}}_{\mathsf{P}}$ are random, by Game 5 the game responds to Z messages to $\mathsf{P}^{\mathsf{sid}}$ as the game in Figure 4.6, and after the modification in oracle H done in Game 7 this oracle also acts as in Figure 4.6. This completes the argument that the real-world and the ideal-world interactions

are indistinguishable to the environment, and hence completes the proof of Theorem 4.1. □

4.2.2 One-Pass HMQV as key-hiding one-time-key AKE

Similiarly to the case of 3DH, we show that a one-pass version of the HMQV protocol [100, 78] realizes functionality $\mathcal{F}_{\mathsf{otkAKE}}$ under the same Gap CDH assumption in ROM. HMQV is a significantly more efficient AKE protocol compared to 3DH because it replaces 3 variable-base exponentiations with 1 multi-exponentiation with two bases. Just like 3DH, HMQV involves both the ephemeral sessions secrets (x, y) and the long-term keys (a, b), and computes session key using a DH-like formula $g^{(x+da)\cdot(y+eb)}$ where d and e are derived via an RO hash of the ephemeral DH contributions, resp. $X = g^x$ and $Y = g^y$.

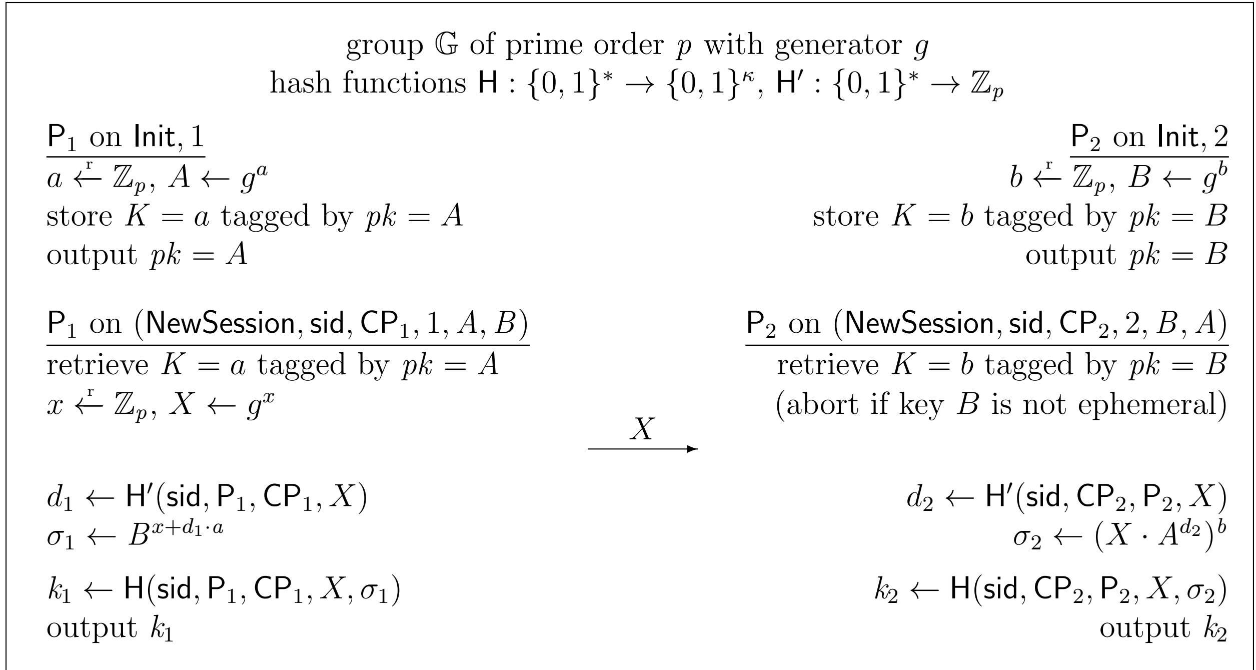


Figure 4.7: otkAKE protocol One-Pass HMQV

Gu et al. [74] showed that HMQV realizes the same key-hiding AKE functionality as 3DH, and here we show that a one-pass HMQV realizes the key-hiding *one-time-key* AKE functionality $\mathcal{F}_{\mathsf{otkAKE}}$. Just like in 2DH, in one-pass HMQV pair (b, B) is a one-time key of party S, which effectively plays the role of both server's public key and its ephemeral DH

contribution. Hence just as 2DH, the only ephemeral DH contribution needed is pair (x, X) provided by C, and the session key can be derived as $g^{(x+a)\cdot b}$. The full protocol is shown in Figure 4.7. As in 2DH we assume that the client and server keys are created before protocol execution, but that the server's key must be a one-time key which is used once and erased afterwards.

We capture the security of one-pass HMQV in the following theorem:

Theorem 4.2. *Protocol One-Pass HMQV shown in Fig 4.7 realizes $\mathcal{F}_{\mathsf{otkAKE}}$ if the Gap CDH assumption holds on group $\mathbb{G}$ and H is a random oracle.*

The proof of theorem 4.2 follows the template of the proof for the corresponding theorem on 2DH security, i.e. Theorem 4.1. It is also a variant of the similar proof shown in [74] which showed that the full HMQV realizes the permanent-key variant of the key-hiding functionality $\mathcal{F}_{\mathsf{otkAKE}}$ defined therein.

Below we show the proof for theorem 4.2. As in the case of 2DH, for each AKE session $\mathsf{C}^{\mathsf{sid}}$(resp. $\mathsf{S}^{\mathsf{sid}}$) we define function $R^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', \perp)$(resp. $R^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', Z)$) which is used to compute its session key given message Z. The definition of $R^{\mathsf{sid}}_{\mathsf{P}}$ is exactly the same as in the case of 2DH, i.e. equation (4.3)(4.4), except the last argument, σ, is now defined using the One-Pass HMQV function, $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$. Below we define function $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ for session $\mathsf{P}^{\mathsf{sid}}$ running on inputs $(\mathsf{sid}, \mathsf{CP}, pk, pk')$, i.e. pk is its own public key and pk' is the public key of the intended counterparty. Function $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ can be defined separately for cases for $\mathsf{P}^{\mathsf{sid}}$ playing the client-role, denoted $\mathsf{P} = \mathsf{C}$, and $\mathsf{P}^{\mathsf{sid}}$ playing the server-role, denoted $\mathsf{P} = \mathsf{S}$. Let $\mathsf{st} = \mathsf{sid}|\mathsf{C}|\mathsf{S}$, the definition is as follows:

$$\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', \perp) = \quad \mathsf{cdh}_g(pk', X) \cdot \mathsf{cdh}_g(pk, pk')^d \text{ for } X = X^{\mathsf{sid}}_{\mathsf{C}}, d = \mathsf{H}'(\mathsf{st}, X)$$

$$\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', Z) = \quad \mathsf{cdh}_g(pk, Z) \cdot \mathsf{cdh}_g(pk, pk')^d \text{ for } d = \mathsf{H}'(\mathsf{st}, Z)$$

Initialization: Initialize empty lists $PK^1_{\mathsf{P}}, PK^2_{\mathsf{P}}, KL_{\mathsf{P}}$ for each P

On message $(\mathsf{Init}, \mathsf{role})$ to P:
pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add pk to $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$:
  if $\mathsf{role} = 1$, pick $w \xleftarrow{r} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K, pk_{\mathsf{CP}}, w\rangle$, output $W = g^w$, set
    $\sigma \leftarrow ({pk_{\mathsf{CP}}}^w \| {pk_{\mathsf{CP}}}^K)$, $k \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, W, \sigma)$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$
  else if $pk_{\mathsf{P}} \in PK^2_{\mathsf{P}}$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K, pk_{\mathsf{CP}}, \perp\rangle$

On message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A, \perp\rangle$:
  set $d \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{S}, \mathsf{C}, Z)$, $\sigma \leftarrow (Z \cdot A^d)^b$, $k \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{S}, \mathsf{C}, Z, \sigma)$
  output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$:
if $\exists\ \langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ in $\mathsf{T_H}$ then output k
else pick $k \xleftarrow{r} \{0,1\}^\kappa$, add $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ to $\mathsf{T_H}$, and output k

On H' query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z)$:
if $\exists\ \langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ in $\mathsf{T_{H'}}$ then output r
else pick $r \xleftarrow{r} \mathbb{Z}_p$, add $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ to $\mathsf{T_{H'}}$, and output r

Figure 4.8: One-Pass HMQV: Environment's view of real-world interaction (Game 0)

Proof. We then give the security proof of One-Pass HMQV which is adjusted from 2DH but simplified. Below we sketch where the match is exact and where the One-Pass-HMQV-specific differences occur and how to deal with them.

GAME 0 *(real world)*: The real-world game is the real world view of executing protocol 4.7.

GAME 1 *(past H queries are irrelevant to new sessions)*: We add an abort if session $\mathsf{C}^{\mathsf{sid}}$ starts with W which appeared in some prior inputs to H. As in the case of 2DH, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq q_{\mathsf{H}}/p$.

GAME 2 *(programming $R^{\mathsf{sid}}_{\mathsf{P}}$ values into H outputs)*: We make the same change of using ran-

Initialization: Initialize an empty list KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P}, \mathsf{role})$ from $\mathcal{F}$:

pick $K \xleftarrow{r} \mathbb{Z}_p$, set $pk \leftarrow g^K$, add (K, pk) to KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:

if $\exists$ $(K, pk) \in KL_{\mathsf{P}}$ send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$, if $\mathcal{F}$ returns "yes" send K to $\mathcal{A}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:

if $\mathsf{role} = 1$: pick $w \xleftarrow{r} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w\rangle$, send $W = g^w$ to $\mathcal{A}$

send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}$

else store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, \perp\rangle$

On $\mathcal{A}$'s message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message counts):

if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, \perp\rangle$:

if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, z\rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$ to $\mathcal{F}$

send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, Z)$ to $\mathcal{F}$

On query (st, σ) to random oracle H, for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X)$:

if $\exists$ $\langle (\mathsf{st}, \sigma), k\rangle$ in $\mathsf{T_H}$ then output k, otherwise pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x\rangle$, $(a, A) \in KL_{\mathsf{C}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X), d\rangle$, in $\mathsf{T_{H'}}$ s.t. $(X, \sigma) = (g^x, B^{x+da})$ for some B:

send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, \perp)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$

if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, \perp\rangle$, $(b, B) \in KL_{\mathsf{S}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X), d\rangle$, in $\mathsf{T_{H'}}$ s.t. $\sigma = (X \cdot A^d)^b)$ for some A:

send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$

add $\langle (\mathsf{st}, \sigma), k\rangle$ to $\mathsf{T_H}$ and output k

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z)$ to random oracle $\mathsf{H'}$:

if $\exists$ $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ in $\mathsf{T_{H'}}$ then output r

else pick $r \xleftarrow{r} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ to $\mathsf{T_{H'}}$, and output r

Figure 4.9: Simulator SIM showing that protocol One-Pass HMQV realizes $\mathcal{F}_{\mathsf{otkAKE}}$

dom but pair-wise correlated functions $R^{\mathsf{sid}}_{\mathsf{P}}$, i.e. correlated as in equation (4.5), and programming $R^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', \perp)$(resp. $R^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', Z)$) values into outputs of $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, W, \sigma)$(resp. $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z, \sigma)$) if Z matches the value sent by $\mathsf{C}^{\mathsf{sid}}$ and $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', \perp)$(resp. $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', Z)$). As in the case of 2DH we need to argue that if the same hash query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$, for $X = X^{\mathsf{sid}}_{\mathsf{C}}$, matches both the client-side equation and the server-side equation, i.e. if

$$\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{C}}(A, B', \perp) = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{S}}(B, A', X)$$

where $A \in PK_{\mathsf{C}}, B' \in PK^{+}(\mathsf{C}^{\mathsf{sid}}), B \in PK_{\mathsf{S}}, A' \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$, as defined in the 2DH proof, then either condition programs the same value into H output.

In the case of 2DH the corresponding equation implied that both parties must use correct counterparty keys, i.e. that $(A', B') = (A, B)$, in which case constraint (4.5) on $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ implies that either condition programs H to the same value.

In the case of One-Pass HMQV the above equation can hold even if $(A', B') \neq (A, B)$, but it can occur with only negligible probability. The constraint above implies:

$$B'^{x+da} = (X \cdot A'^{d})^{b} \tag{4.6}$$

where $d = \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X)$ and $X = g^{x}$. Note that equation (4.6) holds if and only if $b'(1 + a) = b(1 + a')$, where a', b' are the discrete logarithms of resp. A', B'. This can hold even if $(a', b') \neq (a, b)$, hence in the case of One-Pass HMQV we will add an abort in the case equation (4.6) holds and $(A', B') \neq (A, B)$. Note that the adversary must choose the counterparty key $pk' = B'$ for session $\mathsf{C}^{\mathsf{sid}}$ before $\mathsf{C}^{\mathsf{sid}}$ starts and picks x. Likewise $pk' = A'$ for session $\mathsf{S}^{\mathsf{sid}}$ must be chosen before $\mathsf{S}^{\mathsf{sid}}$ starts and picks b, since S needs to pick b randomly everytime it starts a new session. Therefor the last value to be picked is either x or b, i.e. either x or b is randomly sampled after (a, b', a') are all fixed. If x is chosen after (a, b, a', b') then its choice determines $d = \mathsf{H}'(\mathsf{st}, 1, g^{x})$, but since H' is a random oracle, the probability that d satisfies equation (4.6) is $1/p$. If b is chosen after (a, a', b', x) the probability that b satisfies equation (4.6) is $1/2^{\kappa}$. It follows that:

$$|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq q_{\text{ses}}/p + q_{\text{ses}}/2^{\kappa}$$

Game 3 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: This step is identical as in the case of 2DH, and $\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$

GAME 4 *(abort on* H *queries for passive sessions)*: As in the case of the proof for 2DH we add an abort whenever adversarial H queries trigger evaluation in passive sessions(as defined in 2DH proof), which evaluate (1) $R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', \perp)$ or (2)$R_{\mathsf{S}}^{\mathsf{sid}}(pk', pk, Z)$ for any $pk \in PK_{\mathsf{S}}^1$, any $pk' \in PK_{\mathsf{S}}^2$ and $Z = W_{\mathsf{C}}^{\mathsf{sid}}$ where $\mathsf{C}^{\mathsf{sid}}$ is the matching session of $\mathsf{S}^{\mathsf{sid}}$, and likewise define as Bad the event that such query is made.

As in the case of 2DH we show that solving Gap CDH can be reduced to causing event Bad in this game. We argue reduction $\mathcal{R}$ assuming that event Bad occurs for a client-side function $R_{\mathsf{C}}^{\mathsf{sid}}$, and the case for a server-side function $R_{\mathsf{S}}^{\mathsf{sid}}$ is symmetric.

Reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{B})$ and sets $X = \bar{X}^s$ for random s and sends X as message on behalf of $\mathsf{C}^{\mathsf{sid}}$ sections. $\mathcal{R}$ also responds to $(\mathsf{Init}, 1)$ by picking all honestly generated key pairs (a, A) and sets each server public key $pk = B = \bar{B}^t$ for random t just like in the 2DH case. Although $\mathcal{R}$ does not know $x = s \cdot \bar{x}$ and $b = t \cdot \bar{b}$ corresponding to X, B, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{b} = \mathsf{dlog}_g(\bar{B})$, reduction $\mathcal{R}$ can use the DDH oracle to emulate H queries, i.e. to test if

$$\sigma = \mathsf{HMQV}_{\mathsf{C}}^{\mathsf{sid}}(A, B, \perp) = B^{x+da} = \mathsf{cdh}_g(X, B) \cdot B^{ad}$$

for any key $A = g^a$ of $\mathsf{C}^{\mathsf{sid}}$, and $X = \bar{X}^s$ sent by $\mathsf{C}^{\mathsf{sid}}$. Symmetrically $\mathcal{R}$ can test if $\sigma = \mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(B, A, X) = \mathsf{cdh}_g(X \cdot A^d, B)$.

Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3, in which case $\mathcal{R}$ can compute $\mathsf{cdh}_g(X, B)$, assuming Bad occurs for a client-side equation. Then $\mathcal{R}$ can consequently solve $\mathsf{cdh}_g(\bar{X}, \bar{B}) = (\mathsf{cdh}_g(X, B))^{1/(st)}$. It follows that $\Pr[\mathsf{Bad}] \leq \epsilon_{\text{g-cdh}}^{\mathcal{Z}}$, hence:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon_{\text{g-cdh}}^{\mathcal{Z}}$$

Game 5 *(random keys on passively observed sessions)*: This game change is the same as in the case of 2DH, and $\Pr[\mathsf{G5}] = \Pr[\mathsf{G3}]$

Game 6 *(decorrelating function pairs $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$)*: This game change is the same as in the case of 2DH, and $\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$

Game 7 *(hash computation consistent only for compromised keys)*: As in the proof for 2DH, we restrict handling H queries to only those that correspond to counterparty key pk' being either compromised or adversarial. Consequently, as in the case of 2DH, Game 7 diverges from Game 6 if event Bad_2 occurs, defined as H query on server side $(\mathsf{sid}, \mathsf{S}, \mathsf{C}, Z, \sigma)$ for $\sigma = \mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', Z)$, where $pk \in PK_{\mathsf{S}}$ and $pk' \in PK \setminus CPK$.

As in the case of the 2DH proof we will focus on sub-event $\mathsf{Bad}_2[i]$ which denotes Bad_2 occurring where $\mathsf{S}^{\mathsf{sid}}$ uses the i-th key as pk', i.e. pk' was a non-compromised public key in PK created in the i-th key initialization query. Note that $\mathsf{Bad}_2[i]$ corresponds to $\sigma = \mathsf{cdh}_g(X \cdot pk'^d, B)$ where X is arbitrary, B is one-time and uncompromisable public key of session $\mathsf{S}^{\mathsf{sid}}$ and pk' equals to the honestly-generated and non-compromised client public key corresponding to the i-th key record.

As in the 2DH proof we show a reduction that solves a Gap *Square* DH if $\mathsf{Bad}_2[i]$ occurs. Square DH is a variant of CDH where the challenge is a single value $\bar{X}$ and the goal is to compute $\mathsf{cdh}_g(\bar{X}, \bar{X})$. It's also well-known that Square DH is equivalent to CDH. Here we will use a subsidiary reduction $\mathcal{R}$ which computes CDH on a problem *related* to the Square DH challenge, and then use a top-level reduction $\mathcal{R}'$ which solves the Square DH challenge using rewinding over two executions of $\mathcal{R}$. We show the bound on the probability that $\mathcal{R}$ succeeds in terms of the probability ϵ of event $\mathsf{Bad}_2[i]$, and then the overall bound using a union bound.

$\mathcal{R}$ takes a Square DH challenge $\bar{B}$, and embeds $\bar{B}$ into each server public key $B \leftarrow \bar{B}^t$ for

Initialization: Initialize empty lists: PK, PK^1_{P}, PK^2_{P}, CPK, and KL_{P} for all P

On message $(\mathsf{Init}, \mathsf{role})$ to P:
set $K \xleftarrow{r} \mathbb{Z}_p$, $pk \leftarrow g^K$, send $(\mathsf{Init}, \mathsf{role}, pk)$ to P, add pk to PK and $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then add pk to CPK and output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K, pk_{\mathsf{P}}) \in KL_P$ then:
initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$
if $\mathsf{role} = 1$, pick $w \xleftarrow{r} \mathbb{Z}_p$, $k \xleftarrow{r} \{0,1\}^\kappa$,
write rec $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k\rangle$ as fresh, output $W = g^w$,
if $pk_{\mathsf{CP}} \notin PK^2_{\mathsf{CP}}$ then mark rec $\mathsf{interfered}$ and set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp)$
send $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P
else write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp, \perp\rangle$ as fresh

On message Z to session $\mathsf{S}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, B, A, \mathsf{role}, w, \perp\rangle$:
if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, A', B', \mathsf{role}', z, k'\rangle$ s.t. $g^z = Z$
then if rec' is fresh, $(B, A) = (B', A')$, and $k' \neq \perp$:
then $k \leftarrow k'$
else $k \xleftarrow{r} \{0,1\}^\kappa$
else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, Z)$ and re-label rec as $\mathsf{interfered}$
update rec to $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, B, A, \mathsf{role}, w, k\rangle$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma)$:
if $\exists\ \langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ in T_{H} then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot\rangle$, $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X), d\rangle$ in $\mathsf{T}_{\mathsf{H'}}$ s.t. $(X, \sigma) = (g^x, (B)^{x+d\cdot a})$ for some $(a, A) \in KL_{\mathsf{C}}$ and B s.t. $B \in CPK$ or $B \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \perp)$

2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, \perp, \cdot\rangle$, $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X), d\rangle$ in $\mathsf{T}_{\mathsf{H'}}$ s.t. $\sigma = (X \cdot A^d)^b$ for some $(b, B) \in KL_{\mathsf{S}}$ and A s.t. $A \in CPK$ or $A \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$

add $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, \sigma), k\rangle$ to T_{H} and output k

On $\mathsf{H'}$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z)$:
if $\exists\ \langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ in $\mathsf{T}_{\mathsf{H'}}$ then output r
else pick $r \xleftarrow{r} \mathbb{Z}_p$, add $\langle(\mathsf{sid}, \mathsf{C}, \mathsf{S}, Z), r\rangle$ to $\mathsf{T}_{\mathsf{H'}}$, and output r

Figure 4.10: One-Pass HMQV: Environment's view of ideal-world interaction (Game 7)

random t. $\mathcal{R}$ picks each protocol message $X = g^x$ for random x of its choice. $\mathcal{R}$ also picks all keys (a, A) as in $\mathsf{G0}$., except for a chosen index $i \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ sets the key generated in the i-th call to $(\mathsf{Init}, 1)$ as $pk[i] \leftarrow \bar{B}$. Let $\mathsf{Bad}_2[i]$ denote event Bad_2 occurring for A which is this i-th key, i.e. $A = \bar{B}$.

As long as key $pk[i]$ is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys, and it can service H queries as follows: To test client-side σ's, i.e. if $\sigma = B^{a \cdot d} \cdot B^x$, reduction $\mathcal{R}$ tests it as Game 6 does except for a that corresponds to the ith public key $A = \bar{B}$, in which case it tests if $\sigma = \mathsf{cdh}_g(A, B)^d \cdot B^x$ using DDH oracle. To test server-side σ's, i.e. if $\sigma = (X \cdot A^d)^b$ for $b = t \cdot \bar{b}$ where $\bar{b} = \mathsf{dlog}_g(\bar{B})$ and $A \in PK^+(\mathsf{S}^{\mathsf{sid}})$, including $pk[i] = \bar{B}$, reduction $\mathcal{R}$ tests if $\sigma = \mathsf{cdh}_g(X, B) \cdot B^{a \cdot d}$ using DDH oracle except for the case that a is the private key corresponding to the i-th public key $pk = \bar{B}$, or A is an adversarial key, in which case $\mathcal{R}$ tests if $\sigma = \mathsf{cdh}_g(X \cdot A^d, B)$.

Note that $\mathsf{Bad}_2[i]$ can happen only before key $pk[i]$ is compromised, so event $\mathsf{Bad}_2[i]$ occurs in the reduction with the same probability as in Game 6. (If $\mathcal{A}$ asks to compromise $pk[i]$ then $\mathcal{R}$ aborts.) $\mathcal{R}$ can detect event $\mathsf{Bad}_2[i]$ because it occurs if H query involves the public key $pk[i] = \bar{B}$ and σ satisfies the server-side equation for this key, in which case $\mathcal{R}$ computes $\sigma_1 = \mathsf{cdh}_g(X \cdot \bar{B}^{d_1}, \bar{B}^t) = \mathsf{cdh}_g(\bar{B}, \bar{B})^{t \cdot d_1} \cdot \mathsf{cdh}_g(X, \bar{B})^t)$, where we use d_1 to denote $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X)$ in the first execution of $\mathcal{R}$. In the second run of $\mathcal{R}$ hits the event for the same X and embeds fresh d_2 into $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X)$ then it computes $\sigma_2 = \mathsf{cdh}_g(\bar{B}, \bar{B})^{t \cdot d_2} \cdot \mathsf{cdh}_g(X, \bar{B})^t$. Since $\mathcal{R}$ knows t, d_1, d_2, it can output $\mathsf{cdh}_g(\bar{B}, \bar{B}) = (\sigma_1/\sigma_2)^{(1/(t(d_1 - d_2))}$. If $\mathcal{R}$ picks index i at random, by the standard rewinding argument, $\mathcal{R}$ succeeds with probability at least $(1/c_{\mathsf{rwnd}})(\epsilon/q_{\mathrm{K}})^2/q_{\mathsf{H}}$. Thus we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq q_{\mathrm{K}} \cdot (c_{\mathsf{rwnd}} \cdot q_{\mathsf{H}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}})^{1/2}$$

which concludes the proof. $\square$

4.2.3 1/2-SKEME as one-time-key AKE

In this section we introduce protocol 1/2-SKEME, which is a one-pass version of SKEME [97], where one side of parties uses one-time and uncompromisable keys. The message they exchange with each other comes from key encapsulation of counterparty's public key. We define OW-PCA security and strong (key-)anonymous of KEM, see definitions 2.7 and 2.8. The security property of 1/2-SKEME is captured in the following theorem, using the "restricted" key-hiding otkAKE functionality defined below:

"Restricted" Key-hiding one-time-key AKE. We define a "restricted" version of key-hiding *one-time-key* Authenticated Key Exchange (otkAKE), where we add restriction that key secrecy is only protected when the keys are generated by honest party, i.e. if party P runs $\mathsf{NewSession}$ on pk_{CP} which is not on the list of pk's created by the Init query to $\mathcal{F}_{\mathsf{rotkAKE}}$, then $\mathcal{F}_{\mathsf{rotkAKE}}$ reveals keys pk_{CP} to the ideal-world adversary (namely the simulator). The corresponding functionality is denoted as $\mathcal{F}_{\mathsf{rotkAKE}}$ and included it in Figure 4.11. We claim that Theorem 4.4 stands if we replace $\mathcal{F}_{\mathsf{otkAKE}}$ with $\mathcal{F}_{\mathsf{rotkAKE}}$.

Theorem 4.3. *Protocol 1/2-SKEME shown in Figure 4.12 realizes $\mathcal{F}_{\mathsf{rotkAKE}}$, assuming that* KEM *is OW-PCA secure and strong (key-)anonymous, and* H *is a random oracle.*

See definitions of OW-PCA security and strong (key-)anonymous at Definition 2.8 2.7.

Below we show the security proof of Theorem 4.3. As in the case of 2DH and One-Pass HMQV, here we define function $\mathsf{HSKEME}_{\mathsf{P}}^{\mathsf{sid}}$ for session $\mathsf{P}^{\mathsf{sid}}$ running on input $(\mathsf{sid}, \mathsf{CP}, pk, pk')$, i.e. pk is its own public key and pk' is the public key of the intended counterparty. For each AKE session $\mathsf{P}^{\mathsf{sid}}$ we also define function $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', f)$ which is used to compute its session key given message f it received, where $f = d$ on client side and $f = c$ on server side. Function $\mathsf{HSKEME}_{\mathsf{P}}^{\mathsf{sid}}$ and $R_{\mathsf{P}}^{\mathsf{sid}}$ can be defined separately for cases for $\mathsf{P}^{\mathsf{sid}}$ playing the client-role, denoted $\mathsf{P} = \mathsf{C}$, and $\mathsf{P}^{\mathsf{sid}}$ playing the server-role, denoted $\mathsf{P} = \mathsf{S}$. We use $e_{\mathsf{P}}^{\mathsf{sid}}$ and $M_{\mathsf{P}}^{\mathsf{sid}}$

PK stores all public keys created in Init; CPK stores all compromised keys;
PK^1_{P} stores P's permanent public keys; PK^2_{P} stores P's ephemeral public keys;

Keys: Initialization and Attacks

On $(\mathsf{Init}, \mathsf{role})$ from P:

If $\mathsf{role} \in \{1, 2\}$ send $(\mathsf{Init}, \mathsf{P}, \mathsf{role})$ to $\mathcal{A}$, let $\mathcal{A}$ specify pk s.t. $pk \notin PK$, add pk to PK and $PK^{\mathsf{role}}_{\mathsf{P}}$, and output (Init, pk) to P. If P is corrupt then add pk to CPK.

On $(\mathsf{Compromise}, \mathsf{P}, pk)$ from $\mathcal{A}$: [*this query must be approved by the environment*]

If $pk \in PK^1_{\mathsf{P}}$ then add pk to CPK.

Login Sessions: Initialization and Attacks

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ from P:

If $pk_{\mathsf{P}} \in PK^{\mathsf{role}}_{\mathsf{P}}$ and there is no prior session record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \cdot \rangle$ then:

- create session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp \rangle$ marked fresh;
- if $\mathsf{role} = 1$ and $pk_{\mathsf{CP}} \notin PK^2_{\mathsf{CP}}$ then re-label this record as $\mathsf{interfered}$;
- initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : \{0,1\}^3 \to \{0,1\}^\kappa$;
- send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, \perp)$ to $\mathcal{A}$ if $pk_{\mathsf{CP}} \in PK$, else send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{CP}})$.

On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$:

If there is session $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \perp \rangle$ marked fresh then change it to $\mathsf{interfered}$.

Login Sessions: Key Establishment

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ from $\mathcal{A}$:

If $\exists$ session record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, \perp \rangle$ then:

- if rec is marked fresh: If $\exists$ record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk_{\mathsf{CP}}, pk_{\mathsf{P}}, \mathsf{role}', k' \rangle$ marked fresh s.t. $\mathsf{role}' \neq \mathsf{role}$ and $k' \neq \perp$ then set $k \leftarrow k'$, else pick $k \xleftarrow{r} \{0,1\}^\kappa$;
- if rec is marked $\mathsf{interfered}$ then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$;
- update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P.

Session-Key Query

On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$ from $\mathcal{A}$:

If $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot, \cdot \rangle$ and $pk' \notin (PK \setminus CPK)$ then send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$.

Figure 4.11: $\mathcal{F}_{\mathsf{rotkAKE}}$: Functionality for "restricted" key-hiding one-time key AKE

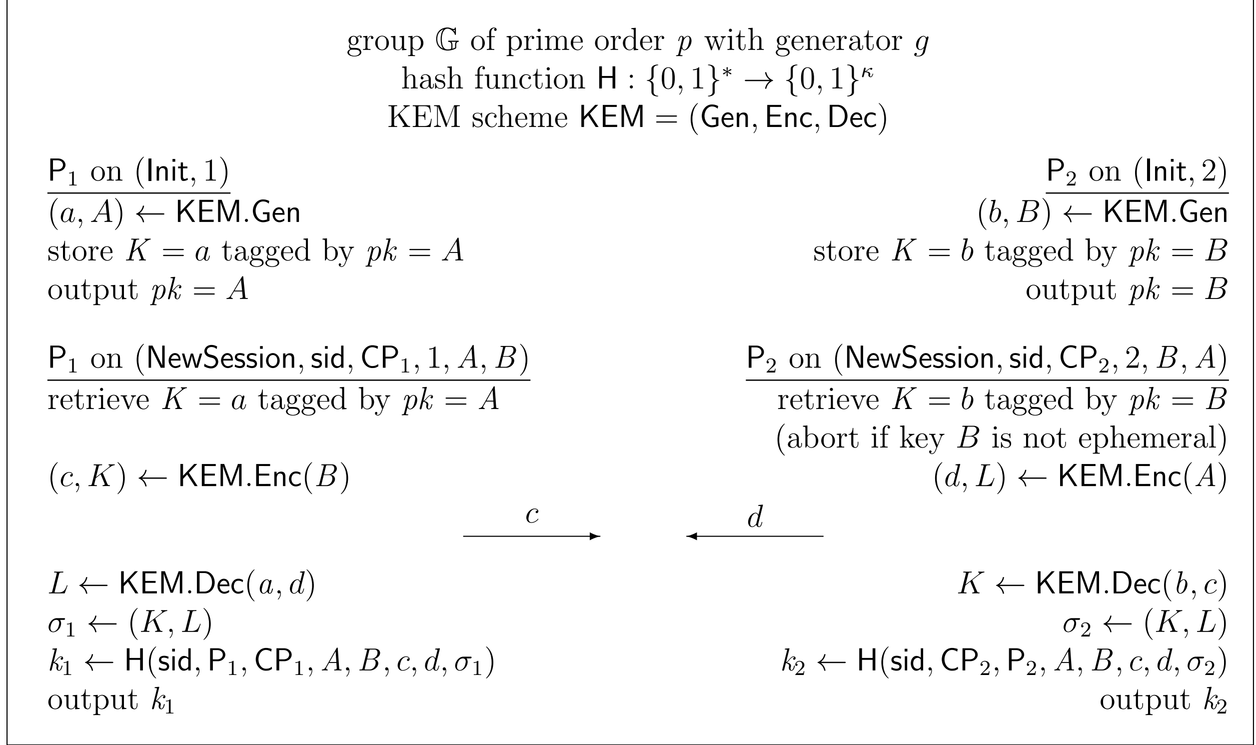


Figure 4.12: otkAKE protocol 1/2-SKEME

to represent (e, M) locally generated via KEM encryption by $\mathsf{P}^{\mathsf{sid}}$ under some $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$. Let $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$, the detailed definitions are as follows[3]:

$$\mathsf{HSKEME}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', d) = (\mathsf{KEM.Dec}(K', c_{\mathsf{C}}^{\mathsf{sid}}), \mathsf{KEM.Dec}(K, d)) \text{ for } (K, pk) \in KL_{\mathsf{C}}$$

$$\text{if } \exists (K', pk') \in KL, \text{ else } (K_{\mathsf{C}}^{\mathsf{sid}}, \mathsf{KEM.Dec}(K, d)) \text{ for } (\cdot, pk') \in KL^{+}(\mathsf{C}^{\mathsf{sid}}) \setminus KL \quad (4.7)$$

$$\mathsf{HSKEME}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', c) = (\mathsf{KEM.Dec}(K, c), \mathsf{KEM.Dec}(K', d_{\mathsf{S}}^{\mathsf{sid}})) \text{ for } (K, pk) \in KL_{\mathsf{S}}$$

$$\text{if } \exists (K', pk') \in KL, \text{ else } (\mathsf{KEM.Dec}(K, c), L_{\mathsf{S}}^{\mathsf{sid}}) \text{ for } (\cdot, pk') \in KL^{+}(\mathsf{S}^{\mathsf{sid}}) \setminus KL \quad (4.8)$$

$$R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', d) = \mathsf{H}(\mathsf{st}, pk, pk', c, d, \mathsf{HSKEME}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', d)) \text{ for } c = c_{\mathsf{C}}^{\mathsf{sid}} \quad (4.9)$$

$$R_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', c) = \mathsf{H}(\mathsf{st}, pk', pk, c, d, \mathsf{HSKEME}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', c)) \text{ for } d = d_{\mathsf{S}}^{\mathsf{sid}} \quad (4.10)$$

[3]Recall that as defined in [74], KL is the list of all key pairs generated so far, and KL_{P} is defined as the set of key pairs generated for P, $KL^{+}(\mathsf{P}^{\mathsf{sid}})$ stands for $KL \cup \{(K_{\mathsf{CP}}, pk_{\mathsf{CP}})\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$ and K_{CP} is corresponding K which doesn't necessarily need to be known or verified. (K', pk') used here are in $KL^{+}(\mathsf{P}^{\mathsf{sid}})$. Note that if $(K_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL$ then $KL^{+}(\mathsf{P}^{\mathsf{sid}}) = KL$.

Initialization: Initialize an empty list KL_{P} for each P

On $(\mathsf{Init}, \mathsf{P}, \mathsf{role})$ from $\mathcal{F}$:
set $(K, pk) \leftarrow \mathsf{KEM.Gen}$, add (K, pk) to KL_{P}, and send pk to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (K, pk) \in KL_{\mathsf{P}}$ send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$ and send K to $\mathcal{A}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, pk)$ from $\mathcal{F}$:
if $pk \neq \perp$ then set $pk^* \leftarrow pk$ else pick $pk^* \stackrel{r}{\leftarrow} PK$
set $(e, M) \leftarrow \mathsf{KEM.Enc}(pk^*)$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, \mathsf{role}, e, M \rangle$, send e to $\mathcal{A}$

On $\mathcal{A}$'s message f to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, \cdot, \cdot, \cdot \rangle$:
  if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, \cdot, f', \cdot \rangle$ s.t. $f = f'$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
  send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, f)$ to $\mathcal{F}$

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$ to random oracle H:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \stackrel{r}{\leftarrow} \{0, 1\}^{\kappa}$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, 1, c, K \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t. $(1) pk = \perp \& \sigma_1 = \mathsf{KEM.Dec}(b, c)$ for $(b, B) \in KL_{\mathsf{S}}$, or $\sigma_1 = K$ for $pk = B$ and $(2) \sigma_2 = \mathsf{KEM.Dec}(a, d)$, send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{C}, A, B, d)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, pk, 2, d, L \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t. $(1) \sigma_1 = \mathsf{KEM.Dec}(b, c)$ and $(2) pk = \perp \& \sigma_2 = \mathsf{KEM.Dec}(a, d)$ for $(a, A) \in KL_{\mathsf{C}}$, or $\sigma_2 = L$ for $pk = A$, send $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{S}, B, A, c)$ to $\mathcal{F}$, if $\mathcal{F}$ returns k^* reset $k \leftarrow k^*$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 4.13: Simulator SIM showing that 1/2-SKEME realizes $\mathcal{F}_{\mathsf{rotkAKE}}$ (abbreviated "$\mathcal{F}$")

Note that $\mathsf{HSKEME}_{\mathsf{P}}^{\mathsf{sid}}$ is defined separately depending on whether pk' is honestly-generated(i.e. on the keylist honest parties generate) or adversarial. $K_{\mathsf{C}}^{\mathsf{sid}}$ and $L_{\mathsf{S}}^{\mathsf{sid}}$ is the plaintext key honestly-generated by running $\mathsf{KEM.Enc}$ on counterparty public key pk' off the list, which are recorded in corresponding session.

We also use σ_1 and σ_2 in the following proof, which supposed to be KEM plaintext key generated by client and server, respectively.

Proof. We now give a security proof of 1/2-SKEME, adjusted from 2DH but simplified, and sketch below where the match is and where the 1/2-SKEME-specific differences occur and how to deal with them.

Initialization: Initialize empty lists $PK^1_{\mathsf{P}}, PK^2_{\mathsf{P}}, KL_{\mathsf{P}}$ for each P

On message $(\mathsf{Init}, \mathsf{role})$ to P:
If $\mathsf{role} \in \{1,2\}$ then set $(K, pk) \leftarrow \mathsf{KEM.Gen}$, add pk to $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}, and output (Init, pk)

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:
if $\exists\ (K_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$, and if $pk_{\mathsf{P}} \in PK^{\mathsf{role}}_{\mathsf{P}}$, then set $(e, M) \leftarrow \mathsf{KEM.Enc}(pk_{\mathsf{CP}})$,
else write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, e, M\rangle$, and output e

On message f to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, K_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, e, M\rangle$:
  set $N \leftarrow \mathsf{KEM.Dec}(K_{\mathsf{P}}, f)$ and $\sigma \leftarrow (M, N)$
  set $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, e, f, \sigma\}_{\mathrm{ord}})$ and output $(\mathsf{sid}, \mathsf{P}, k)$

On H query $(\mathsf{st}, A, B, c, d, \sigma)(\mathsf{st} \leftarrow (\mathsf{sid}, \mathsf{C}, \mathsf{S}))$:
if $\exists\ \langle(\mathsf{st}, A, B, c, d, \sigma), k\rangle$ in $\mathsf{T_H}$ then output k
else pick $k \xleftarrow{r} \{0,1\}^{\kappa}$, add $\langle(\mathsf{st}, A, B, c, d, \sigma), k\rangle$ to $\mathsf{T_H}$, and output k

Figure 4.14: 1/2-SKEME: Environment's view of real-world interaction (Game 0)

GAME 0 *(real world)*: The real-world game is the real world view of executing protocol 4.12.

GAME 1 *(programming $R^{\mathsf{sid}}_{\mathsf{P}}$ values into H outputs)*: We make the same change of using random but pair-wise correlated functions $R^{\mathsf{sid}}_{\mathsf{P}}$, i.e. correlated as in equation (4.9)(4.10), and programming $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', f)$ values into outputs of $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, \{pk, pk', e, f, \sigma\}_{\mathrm{ord}})$ if $\sigma = \mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', f)$, i.e. upon each new query H will respond to hash input in the format of $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$ as follows:

1. If $\exists\ \mathsf{C}^{\mathsf{sid}}$ s.t. $\mathsf{S} = \mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}}$, and $(a, A) \in KL_{\mathsf{C}}$, $(\cdot, B) \in KL^{+}(\mathsf{C}^{\mathsf{sid}})$, and d satisfies $\mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{C}}(A, B, d) = \sigma$, i.e. $(1)\sigma = (\mathsf{KEM.Dec}(b, c^{\mathsf{sid}}_{\mathsf{C}}), \mathsf{KEM.Dec}(a, d))$ if $\exists (b, B) \in KL$ or (2) $\sigma = (K^{\mathsf{sid}}_{\mathsf{C}}, \mathsf{KEM.Dec}(a, d))$ for $(\cdot, B) \in KL^{+}(\mathsf{C}^{\mathsf{sid}}) \setminus KL$, then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, d)$

2. If $\exists\ \mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{C} = \mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}}$, and $(b, B) \in KL_{\mathsf{S}}$, $(\cdot, A) \in KL^{+}(\mathsf{S}^{\mathsf{sid}})$, and c satisfies $\mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{S}}(B, A, c) = \sigma$, i.e. $(1)\sigma = (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Dec}(a, d^{\mathsf{sid}}_{\mathsf{S}}))$ if $\exists (a, A) \in KL$ or (2)

$\sigma = (\mathsf{KEM.Dec}(b, c), L_{\mathsf{S}}^{\mathsf{sid}})$ for $(\cdot, A) \in KL^{+}(\mathsf{S}^{\mathsf{sid}}) \setminus KL$, then set $k \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, c)$

3. In any other case sample $k \xleftarrow{r} \{0,1\}^{\kappa}$

As in the case of 2DH we argue that if the same hash query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$, for $c = c_{\mathsf{C}}^{\mathsf{sid}}$ and $d = d_{\mathsf{S}}^{\mathsf{sid}}$, matches both the client-side equation and the server-side equation, i.e. if

$$\sigma = \mathsf{HSKEME}_{\mathsf{C}}^{\mathsf{sid}}(A, B', d) = \mathsf{HSKEME}_{\mathsf{S}}^{\mathsf{sid}}(B, A', c) \tag{4.11}$$

where $A \in PK_{\mathsf{C}}, B' \in PK^{+}(\mathsf{C}^{\mathsf{sid}}), B \in PK_{\mathsf{S}}, A' \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$, as defined in the 2DH proof, then both conditions program the same value into H output. Denote a, b, a', b' as corresponding secret keys of A, B, A', B', equation (4.11) equals to $(\mathsf{KEM.Dec}(b', c), \mathsf{KEM.Dec}(a, d)) = (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Dec}(a', d))$, which implies that both parties must use correct and honestly generated counterparty keys, i.e. that $(A', B') = (A, B)$, followed by KEM decapsulation security, in which case constraint (4.9)(4.10) on $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ implies that both conditions program H to the same value. Thus we have:

$$|\Pr[\mathsf{G1}] = \Pr[\mathsf{G0}]|$$

GAME 2 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: This step is same as in the case of 2DH, and $\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]$

GAME 3 *(abort on H queries for passive sessions)*: Recall that in the 2DH proof we say a session is passive iff (1)it's a client session that receives honestly-generated ephemeral server public keys and an unmodified server message or (2) it's a server session that runs on honestly-generated client keys and an unmodified client message. Here we add an abort whenever adversarial H queries trigger evaluation in passive sesions, which evaluate (1) $R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', d)$ or (2)$R_{\mathsf{S}}^{\mathsf{sid}}(pk', pk, c)$ for any $(\cdot, pk) \in KL_{\mathsf{C}}$, any $(\cdot, pk') \in KL_{\mathsf{S}}$, $c = c_{\mathsf{C}}^{\mathsf{sid}}$ and $d = d_{\mathsf{C}}^{\mathsf{sid}}$, where

$\mathsf{C}^{\mathsf{sid}}$ is the matching session of $\mathsf{S}^{\mathsf{sid}}$, and likewise define as Bad the event that such query is made. As defined in Game 1, here in passive sessions $R_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', d)$ is triggered only if client side H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, pk', c, d, \sigma)$ satisfies $\sigma = (\mathsf{KEM.Dec}(K', c), \mathsf{KEM.Dec}(K, d))$. The server side is symmetric.

We show a reduction $\mathcal{R}$ that breaks OW-PCA security if Bad occurs. On input a OW-PCA challenge $(\bar{B}, c^*)$, reduction $\mathcal{R}$ has access to $PCO_K(\cdot, \cdot)$ where the inner K corresponds to the private key of $\bar{B}$. $\mathcal{R}$ also picks all key pairs as in Game 0 except for a chosen index $j \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ sets the j-th server public key B as $\bar{B}$, and sets c as c^* in the i-th client session using this server public key. Let $\mathsf{Bad}_{i,j}$ denote Bad occurring for this j-th server public key in the i-th session, i.e. $B = \bar{B}$. Note that all server keys including B are one-time and uncompromisable, and $\mathcal{R}$ can answer any compromise request on client public key A made by adversary by returning corresponding private key a. $\mathcal{R}$ can emulate the way Game 2 services every H queries: to test if a server side H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$ satisfies $\sigma = (K, L)$, $\mathcal{R}$ tests as in Game 1 except for b that corresponds to the public key $\bar{B}$, in which case $\mathcal{R}$ tests K via checking if $PCO_K(K, c)$ returns 1. To test client side queries, i.e. if $\sigma = (K, L)$ for any server public key including $B = \bar{B}$, $\mathcal{R}$ also tests as in Game 1, except for b that corresponds to the public key $\bar{B}$, where $\mathcal{R}$ checks if query $PCO_K(K, c)$ returns 1, including $c = c^*$.

Since $\mathcal{R}$ emulates Game 2 perfectly, event $\mathsf{Bad}_{i,j}$ occurs with the same probability as in Game 2. $\mathcal{R}$ can detect event $\mathsf{Bad}_{i,j}$ because it occurs if H query involves the j-th credential in the session embedded c^* and outputs correct K that satisfies $K = \mathsf{KEM.Dec}(\bar{b}, c^*)$, without knowing the value of $\bar{b}$, in which case it outputs correct K corresponding to c^* and $\bar{B}$, and breaks $OWPCA$ security. If $\mathcal{R}$ picks index i and j at random it follows that $\Pr[\mathsf{Bad}] \leq q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}_{\mathsf{KEM},A}^{ow-pca}$, hence:

$$|\Pr[\mathsf{G3}] - \Pr[\mathsf{G2}]| \leq q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}_{\mathsf{KEM},A}^{ow-pca}$$

Initialization: Initialize empty lists: PK, PK^1_{P}, PK^2_{P}, CPK, KL and KL_{P} for all P

On message $(\mathsf{Init}, \mathsf{role})$ to P:

set $(K, pk) \leftarrow \mathsf{KEM.Gen}$, send (Init, pk) to P, add pk to PK and $PK^{\mathsf{role}}_{\mathsf{P}}$ and (K, pk) to KL_{P}

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:

If $\exists$ $(K, pk) \in KL_{\mathsf{P}}$ and $pk \in PK^1_{\mathsf{P}}$ then add pk to CPK and output K

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, \mathsf{role}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to P:

if $\exists$ $(K, pk_{\mathsf{P}}) \in KL_P$ and $pk_{\mathsf{P}} \in PK^{\mathsf{role}}_{\mathsf{P}}$ then:

  initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$

  if $pk_{\mathsf{CP}} \notin PK$ then set $pk^* \leftarrow pk_{\mathsf{CP}}$, else pick $pk^* \xleftarrow{r} PK$;

  set $(e, M) \leftarrow \mathsf{KEM.Enc}(pk^*)$

  write record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, e, M, \perp \rangle$ as fresh, output e

  if $\mathsf{role} = 1$ and $pk_{\mathsf{CP}} \notin PK^2_{\mathsf{CP}}$ then mark record $\mathsf{interfered}$

On message f to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):

if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, e, M, \perp \rangle$:

  if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \mathsf{role}', f', N, k' \rangle$ s.t. $f = f'$

    then if rec' is fresh, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$ and $k' \neq \perp$:

      then $k \leftarrow k'$

      else $k \xleftarrow{r} \{0,1\}^\kappa$

    else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, f)$ and re-label rec as $\mathsf{interfered}$

  update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, e, M, k \rangle$ and output $(\mathsf{sid}, \mathsf{P}, k)$

On H query $(\mathsf{st}, A, B, c, d, \sigma)(\mathsf{st} \leftarrow (\mathsf{sid}, \mathsf{C}, \mathsf{S}))$:

if $\exists$ $\langle (\mathsf{st}, A, B, c, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output k, else pick $k \xleftarrow{r} \{0,1\}^\kappa$ and:

1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, pk, 1, c, K, \cdot \rangle$, $(a, A) \in KL_{\mathsf{C}}$ s.t.
   $pk = B, \sigma = (K, \mathsf{KEM.Dec}(a, d))$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, d)$

2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, pk, 2, d, L, \cdot \rangle$, $(b, B) \in KL_{\mathsf{S}}$ s.t.
   $(1) pk = \perp \& \sigma = (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Dec}(a, d))$ for $(a, A) \in KL$ s.t. $A \in CPK$ or
   $(2) pk = A, \sigma = (\mathsf{KEM.Dec}(b, c), L)$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, c)$

add $\langle (\mathsf{st}, A, B, c, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output k

Figure 4.15: 1/2-SKEME: Environment's view of ideal-world interaction (Game 7)

Game 4 *(random keys on passively observed sessions)*: This game change is same as in the case of 2DH, and $\Pr[\mathsf{G4}] = \Pr[\mathsf{G3}]$

Game 5 *(decorrelating function pairs $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}}$)*: This game change is the same as in the

case of 2DH, and $\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$

Game 6 *(hash computation consistent only for compromised keys)*: Recall that in Game 5, hash query $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk', pk, e, f, \sigma\}_{\text{ord}})$ is defined as $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', f)$ if $\sigma = \mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{P}}($ $pk, pk', f)$ for some $(\cdot, pk) \in KL_{\mathsf{P}}$ and $(\cdot, pk') \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. In Game 6 we add a condition that this programming of H can occur only if (1) pk' is an adversarial key, i.e. it has not been generated by Init or (2) pk' is an honestly generated permanent key but compromised. These are the two cases the adversary can know K', and we show that they are the only cases where adversary can compute σ s.t. $\sigma = \mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', f)$, and hence trigger the programming of H.

Let $CPKL$ be the list of generated public keys who were compromised so far. Game 6 diverges from Game 5 in the case of event Bad defined as H queried on inputs as above for $\sigma = \mathsf{HSKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, f)$ and $(\cdot, pk_{\mathsf{CP}}) \in KL \setminus CPKL$, i.e. honestly generated and uncompromised keys, while in Game 5, as in Game 1, this programming was done whenever $(\cdot, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$.

We show a reduction $\mathcal{R}$ that breaks $OWPCA$ security if Bad happens on the client side, denoted as Bad_1, and the server side is symmetric. On input a $OWPCA$ challenge $(\bar{B}, c^*)$, $\mathcal{R}$ has access to $PCO_K(.,.)$ whose inner K corresponds to $\bar{B}$. $\mathcal{R}$ also picks all key pairs except that for a chosen index $j \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ set the j-th server public key pk_{CP} as $\bar{B}$, and sets c as c^* in the i-th session using this key, while in other sessions, either use or not use this pk_{CP}, c is generated as previous game. Let $\mathsf{Bad}_{1i,j}$ denote Bad_1 occurring for this j-th server public key in the i-th session, i.e. $pk_{\mathsf{CP}} = \bar{B}$. As long as the corresponding K_{CP} is not compromised, $\mathcal{R}$ can emulate Game 5 because it can respond to compromise of all other keys, and serve H queries as follows: To test server side H query input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$, i.e. if $\sigma = (K, L)$, $\mathcal{R}$ tests as in Game 5 except for b that corresponds to the public key $\bar{B}$, in which case $\mathcal{R}$ tests K via querying $PCO_K(K, c)$ and see if it returns 1, including $c = c^*$. To

test client side hash query, i.e. if $\sigma = (K, L)$ for any pk including $pk = \bar{B}$, $\mathcal{R}$ also tests as in Game 5, except for b that corrsponds to the public key $\bar{B}$, where $\mathcal{R}$ checks if $PCO_K(K, c)$ returns 1 including $c = c^*$.

$\mathsf{Bad}_{1i,j}$ can happen only before pk_{CP} used in that session is compromised, so it occurs in reduction with same probability as in Game 5. $\mathcal{R}$ can detect event $\mathsf{Bad}_{1i,j}$ because it occurs if H query involves the j-th credential and in the i-th client session using this credential it can pass σ check by outputting correct K that satisfies $K = \mathsf{KEM.Dec}(\bar{b}, c^*)$, without knowing the value of $\bar{b}$, which is the secret key corresponding to $\bar{B}$, and thus breaks $OWPCA$ security. If $\mathcal{R}$ picks index i and j at random it follows that $\Pr[\mathsf{Bad}_1] \leq q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$, and since server side is symmetric, we have:

$$|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq 2q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$$

Game 7 *(replace honestly-generated pk_{CP} with randomly-chosen pk^* as* KEM.Enc*'s input)*: The only change we make in this game, is that for all sessions where counterparty public key pk_{CP} is honestly generated via Init, i.e. $pk_{\mathsf{CP}} \in PK$, in $\mathsf{G6}$ in these sessions $(e, M) \leftarrow \mathsf{KEM.Enc}(pk_{\mathsf{CP}})$, in $\mathsf{G7}$ they are changed to be generated via $\mathsf{KEM.Enc}(pk^*)$, where pk^* is randomly chosen from public key space in each such sessions. We argue that this change makes negligible difference to the environment.

Note that for all sessions using some honestly-generated pk_{CP}, the session key k output by P(if not abort) will always be independent from the ciphertext e it generated. In passive sessions, by $\mathsf{G4}$, if party P sets its session key as a random string first, its counterparty will be assigned the same key, or vice versa, no matter ciphertext e is generated by $\mathsf{KEM.Enc}(pk^*)$ or $\mathsf{KEM.Enc}(pk_{\mathsf{CP}})$. Also in sessions where $\mathcal{A}$ interferes with party P then in both games P will be assigned a key $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, f)$, and by $\mathsf{G6}$ result of adversarial hash query could be consistent with k iff $\mathcal{A}$ has compromised pk_{CP}.

We show a reduction $\mathcal{R}$ that breaks strong (key-)anonymous property of KEM[2.7] if the environment can efficiently distinguish this replacement of pk_{CP} in computation of ciphertext e, and we only argue for server side where counterparty client key can be compromised by $\mathcal{A}$, since it's obvious that by prohibiting compromise of server key environment has less advantage in distinguishing this change on client side.

Reduction $\mathcal{R}$ picks all server key pairs and a list of client public keys $[pk_1, pk_2, ..., pk_j, ..., pk_{q_{\mathrm{K}}}]$ and their corresponding list of K. We first define $\mathsf{G6}^{(j,i)}$, where j represents the j-th client public key among the list of public keys $\mathcal{R}$ chooses, and i represents the i-th server session initiated using this j-th client public key. $\mathsf{G6}^{(j,i)}$ acts like $\mathsf{G6}$ but: on all server sessions using client keys $[pk_1, pk_2, ..., pk_{j-1}]$, it generates d from $\mathsf{KEM.Enc}(pk^*)$ (2)on first i server sessions where $pk\mathsf{CP} = pk_j$, we generate d from $\mathsf{KEM.Enc}(pk^*)$. Note that pk^* is randomly chosen in every session mentioned above. It's obvious that $\mathsf{G6}^{(1,0)}$=$\mathsf{G6}$, where on server side for each pk in this list in all sessions $\mathcal{R}$ initiates using pk, d is generated via $\mathsf{KEM.Enc}(pk)$, and $\mathsf{G6}^{(q_{\mathrm{K}},q_{\mathrm{ses}})}$=$\mathsf{G7}$, where q_{ses} is maximum number of sessions initiated using any single pk. And below we show that for every $j \in \{1, ..., q_{\mathrm{K}}\}$, $\mathsf{G6}^{(j,0)}$ is negligibly different from $\mathsf{G6}^{(j,q_{\mathrm{ses}})}$, which indicates $\mathcal{Z}$ cannot notice the difference between $\mathsf{G6}$ and $\mathsf{G7}$. We show this by arguing that for any j set as above, and any $i \in \{1, ..., q_{\mathrm{ses}}\}$ the change between $\mathsf{G6}^{(j,i)}$ and $\mathsf{G6}^{(j,i-1)}$ is negligible, and if environment can notice this change, $\mathcal{R}$ can break key privacy of KEM.

Given a strong (key-anonymous) challenge $(K, pk, K^*, pk^*, \bar{d})$, $\mathcal{R}$ emulates $\mathsf{G6}^{(j,i)}$ like $\mathsf{G6}^{(j,i-1)}$ by embedding pk into pk_j as the j-th client public key and uses K as K_j, and generating ciphertext d via a random key in first $i-1$ sessions using this key, the only difference is that in the i-th session which uses pk_j as pk_{CP}, $\mathcal{R}$ embeds $\bar{d}$ into the ciphertext which this session sends out, while in $\mathsf{G6}^{(j,i-1)}$, the ciphertext is generated via $\mathsf{KEM.Enc}(pk_j)$ in this session. $\mathcal{R}$ can emulate $\mathsf{G6}^{(j,i-1)}$ because it can respond to hash queries in the exact same way: upon $\mathcal{A}$'s server side hash query $\mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, c, d, \sigma)$ where $\sigma = (K, L)$, and d including $\bar{d}$, $\mathcal{R}$ tests as in previous game $\mathcal{A}$ will receive $k = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, c)$ iff $(K, L) =$

$(\mathsf{KEM.Dec}(b,c), \mathsf{KEM.Dec}(a,d))$, where k equals to server side's session key output, and a equals to secret key K_j corresponding to pk_j. In all other cases $\mathcal{A}$ will receive a random key which is independent of the session key.

Then $\mathcal{A}$ outputs a guess bit and $\mathcal{R}$ will use this bit to solve the challenge. Thus we have following equations:

$$\Pr[1 \leftarrow \mathcal{A}|\mathsf{G6}^{(j,i-1)}] = \Pr[1 \leftarrow \mathcal{R}|\bar{d} \leftarrow \mathsf{KEM.Enc}(pk)]$$

$$\Pr[1 \leftarrow \mathcal{A}|\mathsf{G6}^{(j,i)}] = \Pr[1 \leftarrow \mathcal{R}|\bar{d} \leftarrow \mathsf{KEM.Enc}(pk^*)]$$

By definition of strong (key-)anonymous of KEM the probability that $\mathcal{A}$ successfully distinguish $\mathsf{G6}^{(j,i-1)}$ and $\mathsf{G6}^{(j,i)}$ is bounded by $\mathbf{Adv}^{anonym}_{\mathsf{KEM},A}$, which is the adversarial advantage of winning the strong (key-)anonymity game. Since on server side there are totally q_{K} counterparty public keys, and for each public key it can initiate at most q_{ses} sessions, and it's symmetric on client side, we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq 2q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{anonym}_{\mathsf{KEM},A}$$

which concludes the proof. □

4.3 Protocol OKAPE: asymmetric PAKE construction

In this section we show how any UC key-hiding one-time-key AKE protocol can be converted into a UC aPAKE, with very small communication and computational overhead. We call this otkAKE-to-aPAKE compiler OKAPE, which stands for One-time-Key Asymmetric PakE, and we present it in Figure 4.16. As we discussed in the introduction, protocol OKAPE is similar to protocol KHAPE of [74] which is a compiler that creates an aPAKE from any UC

key-hiding AKE where both parties use permanent keys. As in KHAPE, the password file which the server S stores and the password which the client C enters into the protocol, allow them to derive AKE inputs (a, B) for C and (b, A) for S, where (a, A) is effectively a client's password-authenticated public key pair and (b, B) is a server's password-authenticated public key pair, and the authenticated key agreement then consists of executing a key-hiding AKE on the above inputs. (The AKE must be key-hiding or otherwise an attacker could link the keys used by either party to a password they used to derive them.)

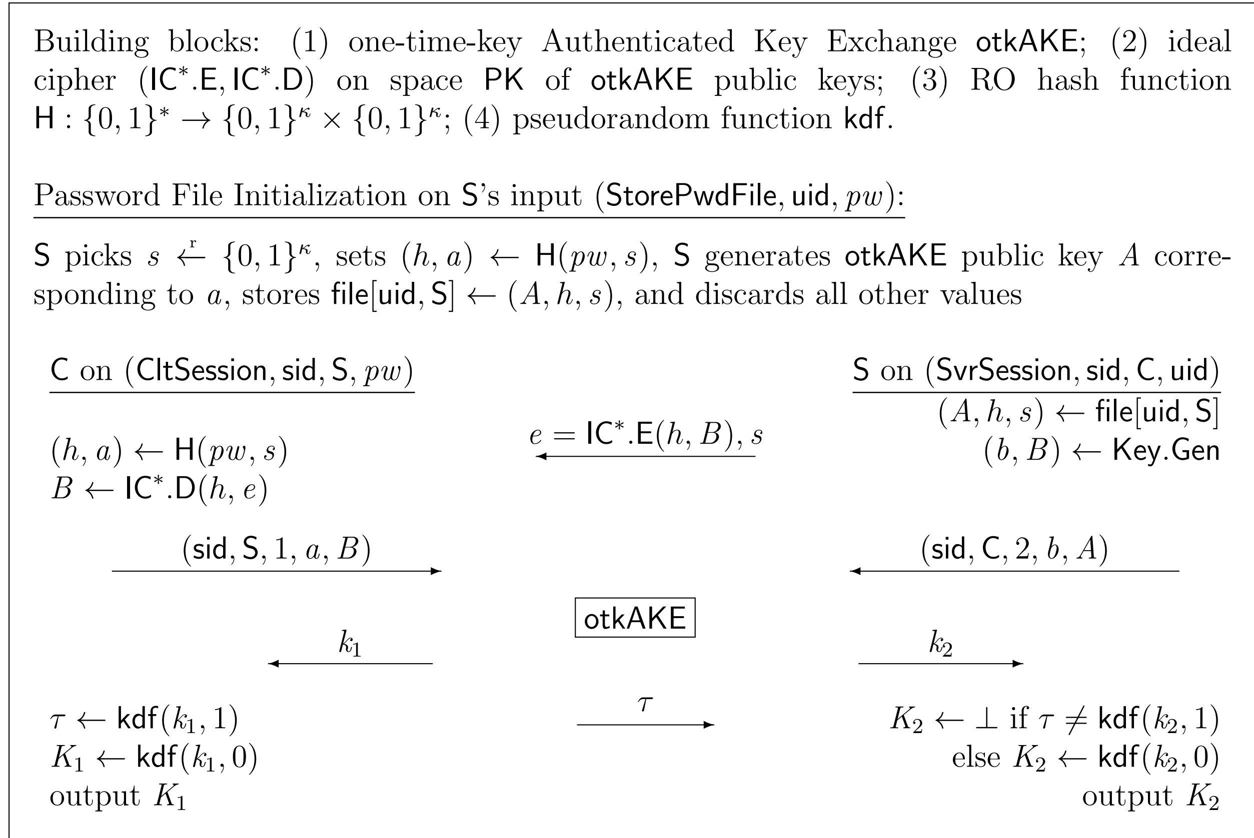


Figure 4.16: Protocol OKAPE: Compiler from key-hiding otkAKE to aPAKE

Protocol otkAKE follows the same general strategy but it differs from KHAPE in (1) how these keys are derived from the client's password and the server's password file, (2) in the type of key-hiding AKE it requires, and (3) whether or not the AKE must be followed by key confirmation messages sent be both parties. In KHAPE the server-side AKE inputs (b, A) were part of the server's password file, and the client-side AKE inputs (a, B) were password-encrypted using an ideal cipher in an envelope $e = \mathsf{IC}.\mathsf{E}_{pw}(a, B)$ stored in the password file

and sent from S to C in each protocol instance. Finally, since both public keys were long-term keys, the protocol required each party to send a key-confirmation message and C needed to send its confirmation before S did or otherwise the protocol would be subject to an offline dictionary attack. The first modification made by OKAPE is that the client's private key a is derived directly as a password hash, and does not need to be encrypted in envelope e. Secondly, there is no permanent server's key (b, B). Instead S generates a *one-time* key pair (b, B) at each protocol instance, and authenticates-and-encrypts its public key B under a password by sending to C an envelope $e = \mathsf{IC.E}_h(B)$ where h is a password hash stored in the password file. Since B is now a one-time key, we can replace key-hiding AKE used in KHAPE with a key-hiding one-time-key AKE, which as we saw in Section 4.2 can be realized with cheaper subprotocols.

More importantly, the IC-encryption of the one-time key B followed by computing the otkAKE session key output by C given input B, implies implicit password-authentication under a unique password: By the properties of the ideal cipher a ciphertext e commits the sender to a single choice of key h (and hence password pw from which h is derived) used to create this ciphertext on a plaintext B chosen by the sender. Hence there can be at most one key h (and thus at most one password pw) s.t. envelope e decrypts to a key B for which the sender knows the corresponding secret key b, and thus can complete the otkAKE protocol ran by C on the key B it decrypts from e. Whereas the protocol still requires a key confirmation by C (otherwise a malicious C could stage an offline dictionary attack once it learned S's session key), the fact that the envelope already implicitly authenticates S implies that it no longer needs a subsequent key confirmation by S.

The main appeal of OKAPE compared to the KHAPE construction in [74] comes from the last implication, i.e. from the fact that we achieve security without the explicit key confirmation from S. If the OKAPE subprotocol is instantiated with either of the two key-hiding otkAKE protocols of Section 4.2, the result is a 2-round aPAKE protocol if S is an initiator and a

3-round protocol if C is an initiator (such concrete instantiation is shown in Figure 3.20 in Section 3.7). Lastly, because S starts the protocol, protocol OKAPE can use (publicly) *salted* password hash at no extra cost to such instantiations: A random salt value s can be part of the password file, the password hash can be defined as $\mathsf{H}(pw, s)$, and s can be delivered from S to C in S's first protocol message, together with envelope e.

This round-complexity improvement is "purchased" at the cost of two trade-offs. First, in OKAPE server S is only implicitly authenticated to C, and if C requires an explicit authentication of S before C uses its session key then the round reduction no longer applies. Secondly, OKAPE can be slightly more computationally expensive than KHAPE because S needs to generate envelope e on-line, which adds an ideal cipher encryption operation to the protocol cost, and current ideal cipher implementations for e.g. elliptic curve group elements have small but non-negligible costs (see Section 3.8).

One additional caveat in protocol OKAPE is that because we want C to derive its AKE private key a from a password hash, we must assume that OKAPE generates private keys from uniformly random bitstrings. This is true about any public key generator if that bitstring is treated as the randomness of the key generator algorithm. For some public key cryptosystems, e.g. RSA, this would be a rather impractical representation of the private key, but in the cryptosystems based on Diffie-Hellman in prime-order groups this randomness can be simply equated with the private key.

Theorem 4.4. *Protocol* OKAPE *realizes the UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *if the AKE protocol realizes functionality* $\mathcal{F}_{\mathsf{otkAKE}}$*, assuming that* kdf *is a secure PRF and* IC^* *is an ideal cipher over the space of* otkAKE *ephemeral public keys.*

Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ is a standard UC aPAKE functionality extended by client-to-server entity authentication. The functionality $\mathcal{F}_{\mathsf{aPAKE}}$ we use is a modification of the UC aPAKE functionality given by [72], but with some refinements we adopt from [74]. This functionality

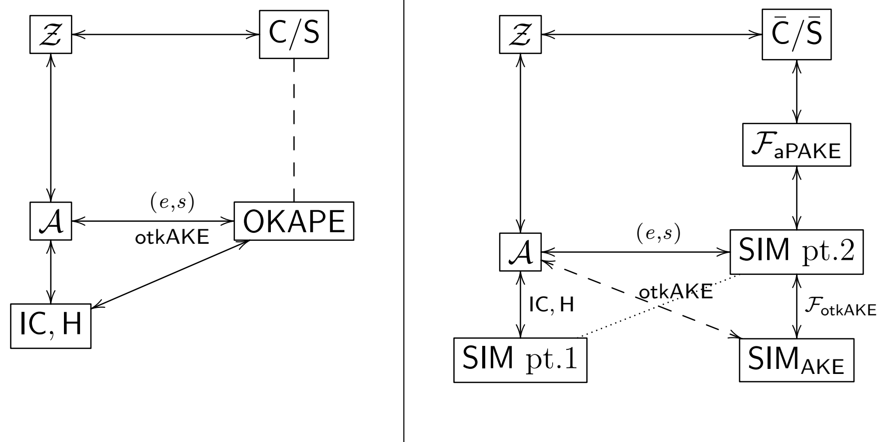


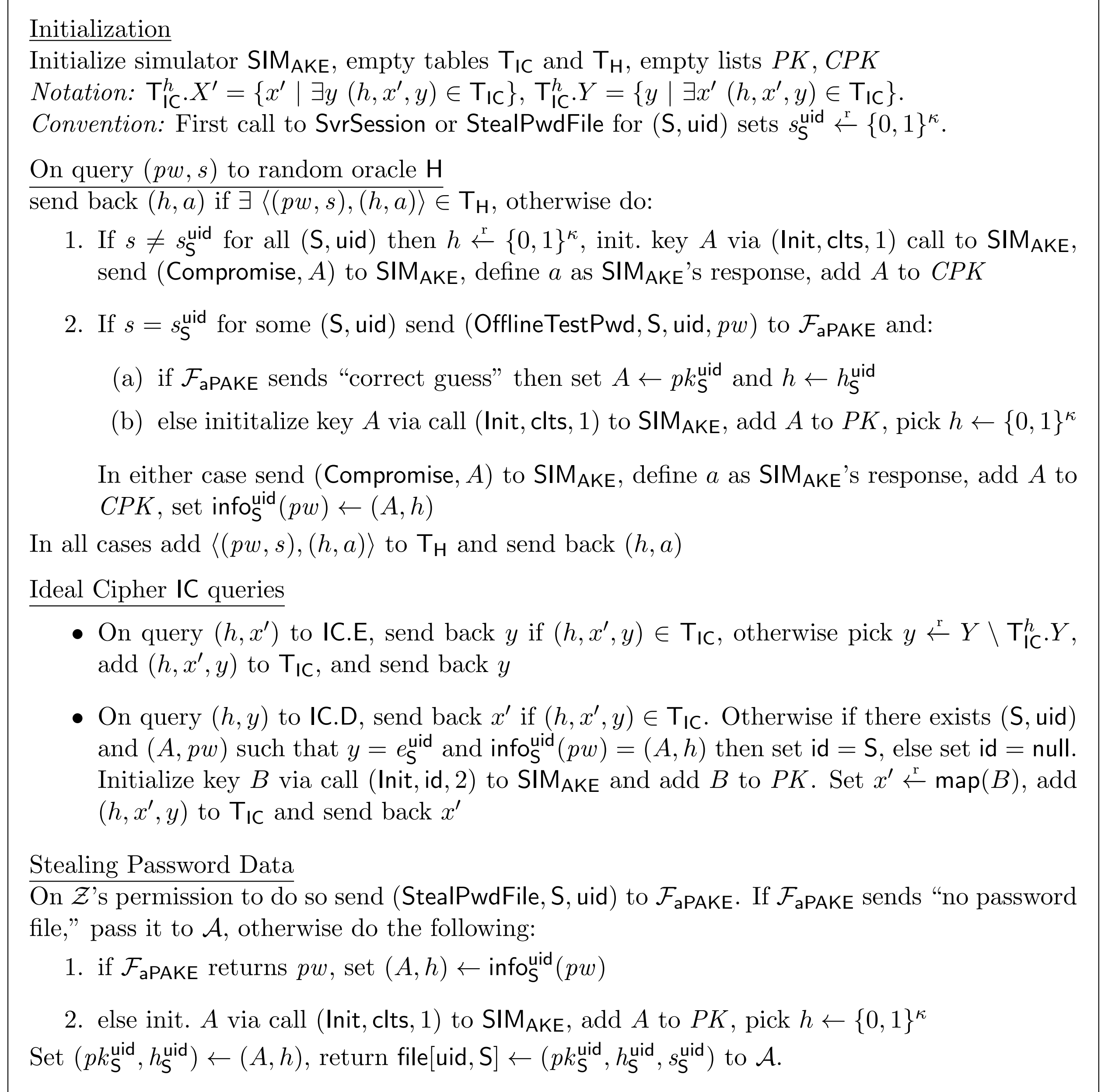
Figure 4.17: real-world (left) vs. simulation (right) for protocol OKAPE

is shown in Fig 2.3.

To prove the theorem, we show that the environment's view of the *real-world* security game, denoted Game 0, i.e. an interaction between the real-world adversary and honest parties who follow protocol OKAPE, is indistinguishable from the environment's view of the *ideal-world* game, denoted Game 10, i.e. an interaction between simulator SIM of Figures 5.20 and 4.19 and functionality $\mathcal{F}_{\mathsf{aPAKE}}$.

Simulator construction. We show an overview of our simulation strategy in Fig 4.17, which gives the top-level view of the real world execution compared to the ideal world execution which involves the simulator SIM shown in Figures 5.20-4.19 as well as the simulator $\mathsf{SIM}_{\mathsf{AKE}}$ for the otkAKE subprotocol. The description of simulator SIM is split into two parts as follows: Figure 5.20 contains the SIM pt.1 part of the diagram in Fig 4.17, i.e. it deals with adversary's ideal cipher and hash queries, and in addition with the compromise of password files. Figure 4.19 contains the SIM pt.2 part of the diagram in Fig 4.17 dealing with on-line aPAKE sessions. We rely on the fact that protocol otkAKE realizes functionality $\mathcal{F}_{\mathsf{otkAKE}}$, so we can assume that there exists a simulator $\mathsf{SIM}_{\mathsf{AKE}}$ which exhibits this UC-security of otkAKE. Our simulator SIM uses simulator $\mathsf{SIM}_{\mathsf{AKE}}$ as a sub-procedure. Namely, SIM hands over to $\mathsf{SIM}_{\mathsf{AKE}}$ the simulation of all C-side and S-side AKE instances where parties run on

either honestly generated or adversarial AKE keys. SIM employs $\mathsf{SIM_{AKE}}$ to generate such keys - in H queries, password file compromise and in IC decryption queries - see Figure 5.20, and then it hands off to $\mathsf{SIM_{AKE}}$ the handling of all AKE instances that run on such keys, see Figure 4.19.



Initialization

Initialize simulator $\mathsf{SIM_{AKE}}$, empty tables $\mathsf{T_{IC}}$ and $\mathsf{T_H}$, empty lists PK, CPK

Notation: $\mathsf{T}^h_{\mathsf{IC}}.X' = \{x' \mid \exists y\ (h, x', y) \in \mathsf{T_{IC}}\}$, $\mathsf{T}^h_{\mathsf{IC}}.Y = \{y \mid \exists x'\ (h, x', y) \in \mathsf{T_{IC}}\}$.

Convention: First call to SvrSession or StealPwdFile for $(\mathsf{S}, \mathsf{uid})$ sets $s^{\mathsf{uid}}_{\mathsf{S}} \xleftarrow{r} \{0,1\}^\kappa$.

On query (pw, s) to random oracle H

send back (h, a) if $\exists\ \langle (pw, s), (h, a) \rangle \in \mathsf{T_H}$, otherwise do:

1. If $s \neq s^{\mathsf{uid}}_{\mathsf{S}}$ for all $(\mathsf{S}, \mathsf{uid})$ then $h \xleftarrow{r} \{0,1\}^\kappa$, init. key A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK
2. If $s = s^{\mathsf{uid}}_{\mathsf{S}}$ for some $(\mathsf{S}, \mathsf{uid})$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and:
   (a) if $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" then set $A \leftarrow pk^{\mathsf{uid}}_{\mathsf{S}}$ and $h \leftarrow h^{\mathsf{uid}}_{\mathsf{S}}$
   (b) else inititalize key A via call $(\mathsf{Init}, \mathsf{clts}, 1)$ to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \leftarrow \{0,1\}^\kappa$

   In either case send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK, set $\mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw) \leftarrow (A, h)$

In all cases add $\langle (pw, s), (h, a) \rangle$ to $\mathsf{T_H}$ and send back (h, a)

Ideal Cipher IC queries

- On query (h, x') to IC.E, send back y if $(h, x', y) \in \mathsf{T_{IC}}$, otherwise pick $y \xleftarrow{r} Y \setminus \mathsf{T}^h_{\mathsf{IC}}.Y$, add (h, x', y) to $\mathsf{T_{IC}}$, and send back y
- On query (h, y) to IC.D, send back x' if $(h, x', y) \in \mathsf{T_{IC}}$. Otherwise if there exists $(\mathsf{S}, \mathsf{uid})$ and (A, pw) such that $y = e^{\mathsf{uid}}_{\mathsf{S}}$ and $\mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw) = (A, h)$ then set $\mathsf{id} = \mathsf{S}$, else set $\mathsf{id} = \mathsf{null}$. Initialize key B via call $(\mathsf{Init}, \mathsf{id}, 2)$ to $\mathsf{SIM_{AKE}}$ and add B to PK. Set $x' \xleftarrow{r} \mathsf{map}(B)$, add (h, x', y) to $\mathsf{T_{IC}}$ and send back x'

Stealing Password Data

On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$. If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "no password file," pass it to $\mathcal{A}$, otherwise do the following:

1. if $\mathcal{F}_{\mathsf{aPAKE}}$ returns pw, set $(A, h) \leftarrow \mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw)$
2. else init. A via call $(\mathsf{Init}, \mathsf{clts}, 1)$ to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \leftarrow \{0,1\}^\kappa$

Set $(pk^{\mathsf{uid}}_{\mathsf{S}}, h^{\mathsf{uid}}_{\mathsf{S}}) \leftarrow (A, h)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (pk^{\mathsf{uid}}_{\mathsf{S}}, h^{\mathsf{uid}}_{\mathsf{S}}, s^{\mathsf{uid}}_{\mathsf{S}})$ to $\mathcal{A}$.

Figure 4.18: Simulator SIM showing that protocol OKAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 1

Notation. We use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i. Hence the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G10}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let

Starting AKE sessions

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$, initialize random function $R^{\mathsf{sid}}_{\mathsf{S}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$, pick $e^{\mathsf{uid}}_{\mathsf{S}} \xleftarrow{r} Y$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $(e^{\mathsf{uid}}_{\mathsf{S}}, s^{\mathsf{uid}}_{\mathsf{S}})$ to $\mathcal{A}$ as a message from $\mathsf{S}^{\mathsf{sid}}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2)$ to $\mathsf{SIM}_{\mathsf{AKE}}$

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$ and message (e', s') sent by $\mathcal{A}$ to $\mathsf{C}^{\mathsf{sid}}$, initialize random function $R^{\mathsf{sid}}_{\mathsf{C}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$, and:

1. If $\exists$ uid s.t. $(e', s') = (e^{\mathsf{uid}}_{\mathsf{S}}, s^{\mathsf{uid}}_{\mathsf{S}})$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$, go to 5.
2. If $\exists$ x', uid s.t. $s' = s^{\mathsf{uid}}_{\mathsf{S}}$ and e' was output by $\mathsf{IC.E}$ on $(h^{\mathsf{uid}}_{\mathsf{S}}, x')$, send $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and in either case below, go to 5:
   (a) If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess", $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow (\mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}, A^{\mathsf{uid}}_{\mathsf{S}}, \mathsf{map}^{-1}(x'))$
   (b) If it returns "wrong guess", set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$.
3. If $\exists$ (x', h, a, pw) s.t. e' was output by $\mathsf{IC.E}$ on (h, x') and $\langle (pw, s'), (h, a) \rangle \in \mathsf{T_H}$ (SIM aborts if tuple duplicated), send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, go to 5 in either case:
   (a) If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess", $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow (\mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}, A, \mathsf{map}^{-1}(x'))$ where A is the public key generated from a.
   (b) If it returns "wrong guess", set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$.
4. In all other cases set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, go to 5.
5. Send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1)$ to $\mathsf{SIM}_{\mathsf{AKE}}$

Responding to $\mathsf{SIM}_{\mathsf{AKE}}$ messages to $\mathcal{F}_{\mathsf{otkAKE}}$ emulated by SIM

SIM passes otkAKE protocol messages between $\mathsf{SIM}_{\mathsf{AKE}}$ and $\mathcal{A}$, but when $\mathsf{SIM}_{\mathsf{AKE}}$ outputs queries to (what $\mathsf{SIM}_{\mathsf{AKE}}$ thinks is) $\mathcal{F}_{\mathsf{otkAKE}}$, SIM reacts as follows:

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$ set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$ and $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$ then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = (\mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}, A, B)$ then $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \alpha)$, output $\tau \leftarrow \mathsf{kdf}(k, 1)$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$
2. Else output $\tau \xleftarrow{r} \{0,1\}^\kappa$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and $\mathcal{A}$ sends τ' to $\mathsf{S}^{\mathsf{sid}}$:

1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by SIM for $\mathsf{C}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$, then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$
2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\exists$ (pw, B) s.t. $\tau' = \mathsf{kdf}(k,1)$ for $k = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, \alpha)$ where $(A, h) = \mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw)$ and $(h, \mathsf{map}(B), e^{\mathsf{uid}}_{\mathsf{S}}) \in \mathsf{T_{IC}}$ (SIM aborts if tuple not unique), send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, pw)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$
3. In any other case send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$:
If $pk' \notin (PK \setminus CPK)$ send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$

Figure 4.19: Simulator SIM showing that protocol OKAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 2

q_{pw}, q_{IC}, q_{H} and q_{ses} be the upper-bounds on the number of resp. password files, IC queries, H queries and online S or C aPAKE sessions. Let $\epsilon^{\mathcal{Z}}_{\text{kdf}}$ and $\epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$ be the advantages of an environment who uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{IC}}+q_{\text{ses}}+q_{\mathsf{pw}})$ exponentiations in $\mathbb{G}$ in resp. breaking the PRF security of kdf, and in distinguishing between the real-world AKE protocol and its ideal-world emulation where $\mathsf{SIM}_{\mathsf{AKE}}$ interacts with $\mathcal{F}_{\mathsf{otkAKE}}$. Let $X' = Y = \{0,1\}^n$ be the domain and range of the ideal cipher IC used within IC*, let X be the domain of public keys in AKE (e.g. for both 2DH and One-Pass HMQV we have $X = \mathbb{G}$ where $\mathbb{G}$ is a group of order p), and let $\mathsf{map} : X \to \{0,1\}^n$ be ϵ_{map}-quasi-bijective.

GAME 0 *(real world)*: This is the interaction, shown in Figure 6.6, of environment $\mathcal{Z}$ with the real-world protocol OKAPE, except that the symmetric encryption scheme is idealized as an ideal cipher oracle and the hash function is idealized as a random oracle. (Technically, this is a hybrid world where each party has access to the ideal cipher functionality IC and to the random oracle H.)

GAME 1 *(embedding private keys in* H *and bookkeeping for honest salts)*: We modify the processing of $\mathcal{Z}$'s query (pw, s) to H for new queries using salts utilized by the protocol, namely when $s = s^{\mathsf{uid}}_{\mathsf{S}}$ for some $(\mathsf{S}, \mathsf{uid})$. We make two changes: (a) instead of picking a randomly, we generate a key-pair (a, A) - note that this is just semantics since we always initialize a private key uniformly - and (b) keep a record $\mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw) \leftarrow (A, h)$ of this query. Clearly $\Pr[\mathsf{G1}] = \Pr[\mathsf{G0}]$.

GAME 2 *(honest salt $s^{\mathsf{uid}}_{\mathsf{S}}$ is never pre-queried)*: We add an abort when a StorePwdFile generates salt $s^{\mathsf{uid}}_{\mathsf{S}}$ for which there is already some pw and (a, h) s.t. $\langle (pw, s^{\mathsf{uid}}_{\mathsf{S}}), (h, a)\rangle \in \mathsf{T}_{\mathsf{H}}$. Since StorePwdFile generates an uniform salt $s^{\mathsf{uid}}_{\mathsf{S}}$, we have $|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq q_{\mathsf{pw}} q_{\mathsf{H}}/2^{\kappa}$.

In the ideal-world game (see Figure 6.11), simulator SIM detects whether the adversary constructed (e', s') honestly - namely by hashing and encrypting his own choice of keys -

Initialize empty table $\mathsf{T_{IC}}$ and $\mathsf{T_H}$; (Notation $\mathsf{T}^h_{\mathsf{IC}}.X'$ and $\mathsf{T}^h_{\mathsf{IC}}.Y$ as in Fig. 5.20)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw^{\mathsf{uid}}_{\mathsf{S}})$ to S: Set $s^{\mathsf{uid}}_{\mathsf{S}} \xleftarrow{r} \{0,1\}^\kappa$, $(h^{\mathsf{uid}}_{\mathsf{S}}, a) \leftarrow \mathsf{H}(pw^{\mathsf{uid}}_{\mathsf{S}}, s^{\mathsf{uid}}_{\mathsf{S}})$, generate public key $A^{\mathsf{uid}}_{\mathsf{S}}$ from a, and set $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (A^{\mathsf{uid}}_{\mathsf{S}}, h^{\mathsf{uid}}_{\mathsf{S}}, s^{\mathsf{uid}}_{\mathsf{S}})$
- On new (pw, s) to H: Pick $(h, a) \xleftarrow{r} (\{0,1\}^\kappa)^2$, add $\langle (pw, s), (h, a) \rangle$ to $\mathsf{T_H}$
- On new (h, x') to $\mathsf{IC.E}$: Output $y \xleftarrow{r} Y \setminus \mathsf{T}^h_{\mathsf{IC}}.Y$, add (h, x', y) to $\mathsf{T_{IC}}$
- On new (h, y) to $\mathsf{IC.D}$: Output $x' \xleftarrow{r} X' \setminus \mathsf{T}^h_{\mathsf{IC}}.X'$, add (h, x', y) to $\mathsf{T_{IC}}$
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Retrieve $(A, h, s^{\mathsf{uid}}_{\mathsf{S}}) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, generate AKE key-pair (b, B), set $e \leftarrow \mathsf{IC.E}(h, \mathsf{map}(B))$, output $(e, s^{\mathsf{uid}}_{\mathsf{S}})$, start AKE session $\mathsf{S}^{\mathsf{sid}}$ on input $(\mathsf{sid}, \mathsf{C}, 2, b, A)$, set k_2 as $\mathsf{S}^{\mathsf{sid}}$ output;

  If $\mathcal{Z}$ sends $\tau' = \mathsf{kdf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$, else set $K_2 \leftarrow \perp$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message (e', s') to C: Set $(h, a) \leftarrow \mathsf{H}(pw, s')$, set $B \leftarrow \mathsf{map}^{-1}(\mathsf{IC.D}(h, e'))$, start AKE session $\mathsf{C}^{\mathsf{sid}}$ on input $(\mathsf{sid}, \mathsf{S}, 1, a, B)$, set k_1 as $\mathsf{C}^{\mathsf{sid}}$ output, set $K_1 = \mathsf{kdf}(k_1, 0)$ and send $\tau = \mathsf{kdf}(k_1, 1)$ to $\mathcal{Z}$;

Figure 4.20: Game 0: $\mathcal{Z}$'s interaction with real-world protocol OKAPE

and extracts the password that the adversary used. We need this password to be unique so we can test it against the actual client password. Therefore we add two aborts so that this detection is unambiguous:

GAME 3 *(abort on ambiguous envelopes)*: We add an abort on IC collisions, i.e., if $\mathsf{IC.Enc}(h_1, x_1) = \mathsf{IC.Enc}(h_2, x_2)$ for two distinct (h_i, x_i). It follows that $|\Pr[\mathsf{G3}] - \Pr[\mathsf{G2}]| \leq q^2_{\mathsf{IC}}/2^{n+1}$.

GAME 4 *(abort on partial* H *collision)*: We also abort if H has a partial collision in the first component of the output, i.e., there exists h and distinct $(pw_i, s_i, a_i)_{i \in \{0,1\}}$ such that $\mathsf{H}(pw_i, s_i) = (h, a_i)$. It follows that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq q^2_{\mathsf{H}}/2^{\kappa+1}$.

GAME 5 *(randomizing the envelope on* SvrSession *initialization)*: We randomize the envelope e created during SvrSession queries: instead of invoking IC.Enc, we pick $e \xleftarrow{r} \{0,1\}^n$, then generate AKE key-pair (b, B), set $x' \xleftarrow{r} \mathsf{map}(B)$ and add (h, x', e) to $\mathsf{T_{IC}}$. We abort if e

was already in T_{IC}, i.e., there exists a h' such that $e \in \mathsf{T}_{\mathsf{IC}}^{h'}.Y$, or if (h, x') had already been queried, i.e., if $x' \in \mathsf{T}_{\mathsf{IC}}^{h}.X'$. But this happens with negligible probability:

$$|\Pr[\mathsf{G5}] - \Pr[\mathsf{G4}]| \leq q_{\mathsf{SvrSession}} \left[\epsilon_{\mathsf{map}} + 2\frac{q_{\mathsf{IC}}}{2^n}\right]$$

Initialize empty table T_{IC} and T_{H}; (Notation $\mathsf{T}_{\mathsf{IC}}^{h}.X'$ and $\mathsf{T}_{\mathsf{IC}}^{h}.Y$ as in Fig. 5.20)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Set$^{(!)}$ $s_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \{0,1\}^{\kappa}$, $(h_{\mathsf{S}}^{\mathsf{uid}}, a) \leftarrow \mathsf{H}(pw_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$, generate public key $A_{\mathsf{S}}^{\mathsf{uid}}$ from a. Store $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$
- On new$^{(!)}$ (pw, s) to H: Pick $h \xleftarrow{r} \{0,1\}^{\kappa}$ and generate AKE key pair (a, A). If there exists $(\mathsf{S}, \mathsf{uid})$ such that $s = s_{\mathsf{S}}^{\mathsf{uid}}$ then record $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, h)$. Either way, add $\langle (pw, s), (h, a) \rangle$ to T_{H} and output (h, a).
- On new$^{(!)}$ (h, x') to $\mathsf{IC.E}$: Output $y \xleftarrow{r} Y \setminus \mathsf{T}_{\mathsf{IC}}^{h}.Y$, add (h, x', y) to T_{IC}
- On new$^{(!)}$ (h, y) to $\mathsf{IC.D}$: Output $x' \xleftarrow{r} X' \setminus \mathsf{T}_{\mathsf{IC}}^{h}.X'$, add (h, x', y) to T_{IC}
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Retrieve $(A, h, s) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, generate AKE key-pair (b, B), set$^{(*)}$ $e \xleftarrow{r} \{0,1\}^{n}$, set $x' \xleftarrow{r} \mathsf{map}(B)$ and add$^{(*)}$ (h, x', e) to T_{IC}. Output (e, s), start AKE session $\mathsf{S}^{\mathsf{sid}}$ on input $(\mathsf{sid}, \mathsf{C}, 2, b, A)$, set k_2 as $\mathsf{S}^{\mathsf{sid}}$ output;

  If $\mathcal{Z}$ sends $\tau' = \mathsf{kdf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$, else set $K_2 \leftarrow \bot$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message (e', s') to C: Set $(h, a) \leftarrow \mathsf{H}(pw, s')$, set $B \leftarrow \mathsf{map}^{-1}(\mathsf{IC.D}(h, e'))$, start AKE session $\mathsf{C}^{\mathsf{sid}}$ on input $(\mathsf{sid}, \mathsf{S}, 1, a, B)$, set k_1 as $\mathsf{C}^{\mathsf{sid}}$ output, set $K_1 = \mathsf{kdf}(k_1, 0)$ and send $\tau = \mathsf{kdf}(k_1, 1)$ to $\mathcal{Z}$;

Figure 4.21: OKAPE Game 5: changes before replacement of protocol with the functionality

Remarks. The current game is shown in Figure 4.21. For notational brevity, we say queries to oracles H, $\mathsf{IC.E}$, and $\mathsf{IC.D}$ are new$^{(!)}$ as a shortcut for saying that the respective table includes no prior tuple corresponding to the query's input. If such tuple exists then the oracle just retrieves the answer from its table. We also omit the possibilities of the game aborting, because such aborts happen only with negligible probability: the places where they could happen are marked$^{(*)}$, and correspond to the aborts that we described in the previous games.

Game 6 *(replacing the* otkAKE *protocol by functionality* $\mathcal{F}_{\mathsf{otkAKE}}$*)*: In this step we replace our protocol by interactions between $\mathcal{F}_{\mathsf{otkAKE}}$ and a simulator $\mathsf{SIM}_{\mathsf{AKE}}$ for the protocol, as is usually done in the UC framework. In particular, we replace all AKE key generations to calls to the simulator $\mathsf{SIM}_{\mathsf{AKE}}$, shown in Figure 4.22, and the game emulates how $\mathcal{F}_{\mathsf{otkAKE}}$ would respond to $\mathsf{SIM}_{\mathsf{AKE}}$'s actions. Note that key-generation is completely delegated to $\mathsf{SIM}_{\mathsf{AKE}}$[4] , and private keys are, w.l.o.g, always chosen uniformly random as in the previous game. When generating a key for the client, we use a special flag clts to denote the fact that this key is not tied to a user, and in fact can be obtained by anyone if they query H with the correct (pw, s) input. This is needed because of deficiencies in the way we handle keys in the UC framework, so that, technically, we have an intermediary game where we replace the runs of the real otkAKE protocol with run with the same identity clts for every client.

Besides delegating key-generation to $\mathsf{SIM}_{\mathsf{AKE}}$, we need to correctly simulate the functionality $\mathcal{F}_{\mathsf{otkAKE}}$ to it, so that the simulator can behave correctly. The precise steps of this game change are described next: they amount to replacing the protocol with the functionality as is usually done in the UC framework.

We initialize a random function $R_{\mathsf{P}}^{\mathsf{sid}}$ for every AKE session $\mathsf{P}^{\mathsf{sid}}$ invoked by emulated $\mathcal{F}_{\mathsf{otkAKE}}$, and sends $\mathsf{NewSession}$ messages to $\mathsf{SIM}_{\mathsf{AKE}}$ whenever C or S starts a session under such generated keys. When the environment behaves adversarially, we simulate $\mathcal{F}_{\mathsf{otkAKE}}$ by marking sessions as actively attacked. In fact, on client side an attack can happen as soon as the client receives its initial input (e, s). If (e, s) is not decrypted to a key created by $\mathsf{SIM}_{\mathsf{AKE}}$ for S - namely created during a $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ query - the functionality $\mathcal{F}_{\mathsf{otkAKE}}$ would mark this session as $\mathsf{interfered}$. Note that a client session $\mathsf{C}^{\mathsf{sid}}$ that receives an honest server generated envelope $(e^{\mathsf{sid}'}_{\mathsf{S,uid}}, s^{\mathsf{uid}}_{\mathsf{S}})$ aimed at a session $\mathsf{sid}' \neq \mathsf{sid}$ will not be $\mathsf{interfered}$, since client

[4]This is a major difference between the KHAPE compiler [74] and the current one: there an adversary could make a client run on arbitrarily chosen private key a, and in that case we can't rely on the security of the khAKE protocol since $\mathcal{F}_{\mathsf{khAKE}}$ does not handle this situation, but here the adversary can only make C run on an honestly (meaning by $\mathsf{SIM}_{\mathsf{AKE}}$) generated key a, which is initialized during the corresponding H query. Even though this private key may be eventually compromised and may have nothing to do with the key that the honest S intended for C to use, $\mathcal{F}_{\mathsf{otkAKE}}$ still handles these cases securely.

Initialize simulator $\mathsf{SIM_{AKE}}$, empty tables $\mathsf{T_{IC}}, \mathsf{T_H}$, and lists CPK, PK.

- On new$^{(!)}$ (pw, s) to H: Pick $h \xleftarrow{r} \{0,1\}^\kappa$. Then init. key A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK. If $s = s_\mathsf{S}^\mathsf{uid}$ for some $(\mathsf{S}, \mathsf{uid})$ then record $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) \leftarrow (A, h)$. Add $\langle (pw, s), (h, a) \rangle$ to $\mathsf{T_H}$ and output (h, a).

- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Retrieve $(A, h, s) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$. Initialize $B_\mathsf{S}^\mathsf{sid}$ via an $(\mathsf{Init}, \mathsf{S}, 2)$ call to $\mathsf{SIM_{AKE}}$, let $e_\mathsf{S}^\mathsf{uid} \xleftarrow{r} \{0,1\}^n$, add $B_\mathsf{S}^\mathsf{sid}$ to PK, let $x' \xleftarrow{r} \mathsf{map}(B_\mathsf{S}^\mathsf{sid})$ and add $(h_\mathsf{S}^\mathsf{uid}, x', e_\mathsf{S}^\mathsf{uid})$ to $\mathsf{T_{IC}}$. Output $(e_\mathsf{S}^\mathsf{uid}, s)$, initialize function $R_\mathsf{S}^\mathsf{sid}$, set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{hbc}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2)$ to $\mathsf{SIM_{AKE}}$

- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and (e, s) to C: Set $(h, a) \leftarrow \mathsf{H}(pw, s)$, generate A corresponding to a, set $B \leftarrow \mathsf{map}^{-1}(\mathsf{IC.D}(h, e))$, initialize function $R_\mathsf{C}^\mathsf{sid}$ and then

  1. if $\exists$ $(\mathsf{uid}, \mathsf{sid}')$ such that $(e, s) = (e_{\mathsf{S},\mathsf{uid}}^{\mathsf{sid}'}, s_\mathsf{S}^\mathsf{uid})$ and $pw = pw_\mathsf{S}^\mathsf{uid}$: set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{hbc}(\mathsf{sid}', A, B)$
  2. else set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{act}(A, B)$

  In either case send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{id}, \mathsf{S}, 1)$ to $\mathsf{SIM_{AKE}}$

Responding to AKE messages:

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{act}$

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(\mathsf{sid}', A, B)$ then change it to $\mathsf{act}(A, B)$

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:

  1. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{act}(A, B)$ set $k_1 \leftarrow R_\mathsf{C}^\mathsf{sid}(A, B, \alpha)$
  2. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(\mathsf{sid}', A, B)$: if $\mathsf{sid} = \mathsf{sid}'$, $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$, and S outputted k_2, then pick $k_1 \leftarrow k_2$, otherwise $k_1 \xleftarrow{r} \{0,1\}^\kappa$

  Output $K_1 \leftarrow \mathsf{kdf}(k_1, 0)$ and $\tau \leftarrow \mathsf{kdf}(k_1, 1)$

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to S^sid:

  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}$, set $k_2 \leftarrow R_\mathsf{S}^\mathsf{sid}(B_\mathsf{S}^\mathsf{sid}, A_\mathsf{S}^\mathsf{uid}, \alpha)$
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$: If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(\mathsf{sid}, A, B)$ and C^sid outputted key k_1 then copy this k_1 to k_2, otherwise pick $k_2 \xleftarrow{r} \{0,1\}^\kappa$

  If $\tau' = \mathsf{kdf}(k_2, 1)$ output $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$, else output $\perp$

- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: output $R_\mathsf{P}^\mathsf{sid}(pk, pk', \alpha)$ if $pk' \notin (PK \setminus CPK)$

Figure 4.22: OKAPE Game 6: replacing the protocol with the functionality $\mathcal{F}_\mathsf{otkAKE}$

will decrypt e to a server generated key (albeit not the one $\mathsf{S}^{\mathsf{sid}}$ uses). In this mismatching sid case, $\mathcal{F}_{\mathsf{otkAKE}}$ would decouple the client key from the server key and consequently the resulting keys won't match. We solve this problem by keeping track of sid'.

If $\mathsf{SIM}_{\mathsf{AKE}}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ to an actively attacked session, the output key k is set to $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the keys this session runs under, i.e. $(B^{\mathsf{uid}}_{\mathsf{S}}, A^{\mathsf{uid}}_{\mathsf{S}})$ for S, and keys (A, B) derived from $\mathsf{H}(pw, s)$ and $\mathsf{IC.D}(h, e)$ for C. The game initializes PK which contains all public keys generated by $\mathsf{SIM}_{\mathsf{AKE}}$, and CPK which contains the subset of permanent keys (equivalently, those that are compromised and need to be stored in the hash function's table). Then the game emulates the $\mathsf{ComputeKey}$ interface of $\mathcal{F}_{\mathsf{otkAKE}}$ and lets $\mathsf{SIM}_{\mathsf{AKE}}$ evaluate $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ for any $pk' \notin (PK \setminus CPK)$. When $\mathsf{SIM}_{\mathsf{AKE}}$ sends NewKey to a non-attacked session, the game emulates $\mathcal{F}_{\mathsf{otkAKE}}$ by issuing uniform keys and we make sure they match between counterparties in the same session iff they run on agreeing keys (equivalently, if $\mathsf{sid} = \mathsf{sid}'$ as we explained above). Finally, we still need to consider the correctness of the confirmation message τ and our server sessions output $\perp$ if they are incorrect.

Since we are assuming our protocol is UC secure we conclude that this game is indistinguishable from the previous one: $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq \epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$.

Game 7 *(delay usage of password files and ephemeral keys $B^{\mathsf{uid}}_{\mathsf{S}}$)*:

In Game 6, key $A^{\mathsf{uid}}_{\mathsf{S}}$, hash $h^{\mathsf{uid}}_{\mathsf{S}}$ and salt $s^{\mathsf{uid}}_{\mathsf{S}}$ are all generated during $\mathsf{StorePwdFile}$ queries, while in our ideal-world they only play an important role after password compromise. In fact, we cannot generate them at the start since the simulator has no knowledge of the password $pw^{\mathsf{uid}}_{\mathsf{S}}$ before the compromise happens. Similarly, in the previous game we generate $B^{\mathsf{uid}}_{\mathsf{S}}$ before it is needed (they are only ever used during actively attacked server sessions) and our goal is to eventually drop its usage just as it happens in our ideal-world Figure 6.11. In Game 5 we begin the process above, and this change will be done in several steps.

The first step is denoted Game 5 (a): we remove the generation of key $B_{\mathsf{S}}^{\mathsf{uid}}$ in $\mathsf{SvrSession}$ query, and instead we delay it to a decryption $x' \leftarrow \mathsf{IC.Dec}(h_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ in the actively attacked server sessions - note that this is the only place where $B_{\mathsf{S}}^{\mathsf{uid}}$ is used in Game 6, except that $\mathsf{SIM}_{\mathsf{AKE}}$ expects to be called in a key owned by S during an honest $\mathsf{CltSession}$. To make this step indistinguishable, we embed keys in $\mathsf{IC.Dec}$ calls to $(h_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$. Because of the properties of map this change is indistinguishable except with probability $q_{\mathsf{IC}}\left[\epsilon_{\mathsf{map}} + \frac{q_{\mathsf{IC}}}{2^n}\right]$ since we need to abort on collisions (see Figure 4.23). Moreover, $\mathcal{Z}$ cannot notice this change as long as the delayed decryption actually generates a key (namely $e_{\mathsf{S}}^{\mathsf{uid}}$ does not come from a new$^{(!)}$ encryption query $\mathsf{IC.Enc}$), and this happens except with negligible probability $q_{\mathsf{SvrSession}} q_{\mathsf{IC}}/2^n$. In this step we also set $s_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \{0,1\}^\kappa$ at the beginning of $\mathsf{SvrSession}$ if it's not set yet. Note this is only a syntactic change since a $\mathsf{SvrSession}$ only happens after $\mathsf{StorePwdFile}$ and the latter generates $s_{\mathsf{S}}^{\mathsf{uid}}$ in the current game.

In Game 5 (b) we start marking $pw_{\mathsf{S}}^{\mathsf{uid}}$ as fresh when a $\mathsf{StorePwdFile}$ query is made, and instead of using the answer from the H query to obtain $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ for the $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ generation, we use the $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ created by this query.

Finally, in Game 5 (c) we delay $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ generation until password compromise. We change both $\mathsf{StealPwdFile}$ and H simultaneously. We begin by changing H with the description in our ideal-world Figure 6.11 where if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is fresh then we record the offline query attempt. If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is correctly guessed and $\mathsf{compromised}$ then we retrieve $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ generated by $\mathsf{StealPwdFile}$ in the change we describe next. Note that so far $pw_{\mathsf{S}}^{\mathsf{uid}}$ is never marked $\mathsf{compromised}$, so this is only a notational change. We then modify $\mathsf{StealPwdFile}$, again copying over the definition from our ideal world figure, where we mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ $\mathsf{compromised}$ and generate $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ if there's no previous offline query attempt on $pw_{\mathsf{S}}^{\mathsf{uid}}$, else we retrieve it from the record $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ that was created during an H query. It is easy to see that this last change is undetectable too, no matter the order of queries to H and $\mathsf{StealPwdFile}$ due to both generating $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ by the same method. As in the previous game, we add

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid})$ to S: mark $pw_\mathsf{S}^\mathsf{uid}$ as fresh, pick$^{(*)}$ $s_\mathsf{S}^\mathsf{uid} \xleftarrow{r} \{0,1\}^\kappa$, query $\mathsf{H}(pw_\mathsf{S}^\mathsf{uid}, s_\mathsf{S}^\mathsf{uid})$ and set $(A_\mathsf{S}^\mathsf{uid}, h_\mathsf{S}^\mathsf{uid}) \leftarrow \mathsf{info}_\mathsf{S}^\mathsf{uid}(pw_\mathsf{S}^\mathsf{uid})$.

- On new$^{(!)}$ (pw, s) to H:

  1. If $s \neq s_\mathsf{S}^\mathsf{uid}$ for all $(\mathsf{S}, \mathsf{uid})$ then $h \xleftarrow{r} \{0,1\}^\kappa$, init. key A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK
  2. If $s = s_\mathsf{S}^\mathsf{uid}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_\mathsf{S}^\mathsf{uid}$ is $\mathsf{compromised}$ and $pw = pw_\mathsf{S}^\mathsf{uid}$ set $(A, h) \leftarrow (A_\mathsf{S}^\mathsf{uid}, h_\mathsf{S}^\mathsf{uid})$
     (b) Else then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, initialize A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \xleftarrow{r} \{0,1\}^\kappa$

     In both cases (a) and (b), set $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) \leftarrow (A, h)$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$ and define a as its response, add A to CPK

  Add $\langle (pw, s), (h, a) \rangle$ to $\mathsf{T_H}$ and send back (h, a)

- On new$^{(!)}$ (h, y) to $\mathsf{IC.D}$: If $(h, y) = (h_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})$ for some $(\mathsf{S}, \mathsf{uid})$ then we Initialize B via $(\mathsf{Init}, \mathsf{S}, 2)$ call to $\mathsf{SIM_{AKE}}$, add B to PK, set $x' \xleftarrow{r} \mathsf{map}(B)$. Otherwise set $x' \xleftarrow{r} X' \backslash \mathsf{T}_\mathsf{IC}^h.X'$. Either way, add (h, x', y) to $\mathsf{T_{IC}}$, output x'

- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: if $s_\mathsf{S}^\mathsf{uid}$ is not set, pick $s_\mathsf{S}^\mathsf{uid} \xleftarrow{r} \{0,1\}^\kappa$. If there is a fresh $pw_\mathsf{S}^\mathsf{uid}$, mark it $\mathsf{compromised}$ and continue, otherwise abort. Then (a) If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid} \rangle$ then set $(A, h) \leftarrow \mathsf{info}_\mathsf{S}^\mathsf{uid}(pw)$; (b) else initialize A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \xleftarrow{r} \{0,1\}^\kappa$.

  In either case, set $(A_\mathsf{S}^\mathsf{uid}, h_\mathsf{S}^\mathsf{uid}) \leftarrow (A, h)$, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (A_\mathsf{S}^\mathsf{uid}, h_\mathsf{S}^\mathsf{uid}, s_\mathsf{S}^\mathsf{uid})$

- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: if $s_\mathsf{S}^\mathsf{uid}$ is not set, pick $s_\mathsf{S}^\mathsf{uid} \xleftarrow{r} \{0,1\}^\kappa$. Initialize function $R_\mathsf{S}^\mathsf{sid}$, set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{hbc}$, let $e_\mathsf{S}^\mathsf{uid} \xleftarrow{r} \{0,1\}^n$, output $(e_\mathsf{S}^\mathsf{uid}, s_\mathsf{S}^\mathsf{uid})$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2)$ to $\mathsf{SIM_{AKE}}$

Responding to AKE messages:

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to S^sid:

  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$: If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}(\mathsf{sid}, A, B)$ and C^sid outputted key k_1 then copy this k_1 to k_2, otherwise pick $k_2 \xleftarrow{r} \{0,1\}^\kappa$
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}$, set $k_2 \leftarrow R_\mathsf{S}^\mathsf{sid}(\mathsf{map}^{-1}(\mathsf{IC.Dec}(h_\mathsf{S}^\mathsf{uid}, e_\mathsf{S}^\mathsf{uid})), A_\mathsf{S}^\mathsf{uid}, \alpha)$

  If $\tau' = \mathsf{kdf}(k_2, 1)$ output $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$, else output $\perp$

Figure 4.23: OKAPE Game 5: delaying password file creation

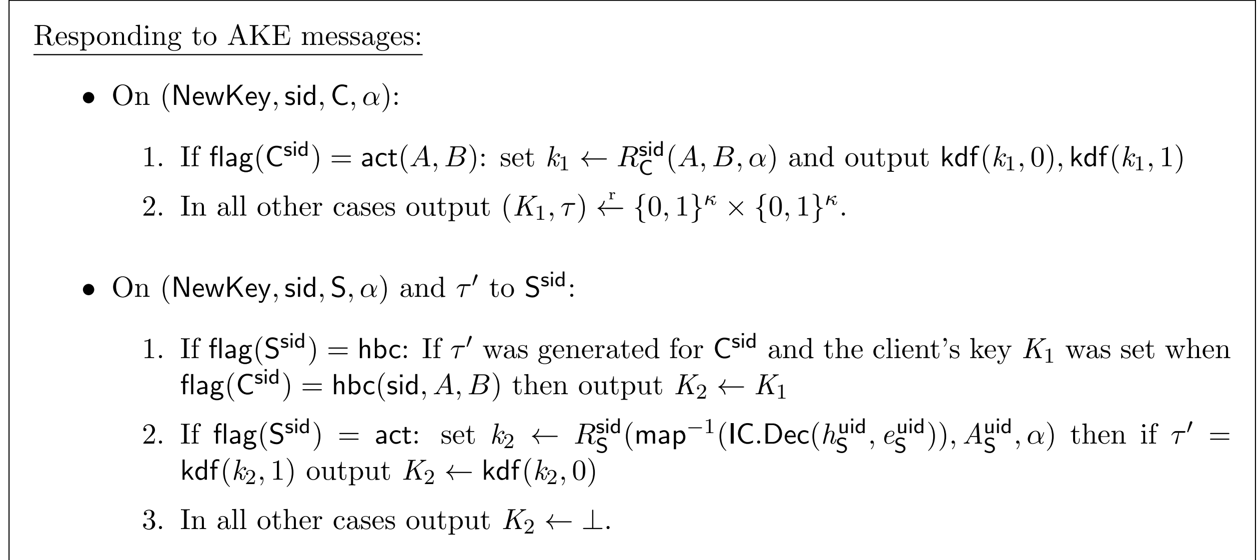
a superfluous generation of $s_{\mathsf{S}}^{\mathsf{uid}}$ so that the next game changes are clearer - note it is still always generated during $\mathsf{StorePwdFile}$. We have $|\Pr[\mathsf{G5}]-\Pr[\mathsf{G6}]| \le q_{\mathsf{IC}}\left[\epsilon_{\mathsf{map}} + \frac{q_{\mathsf{IC}}}{2^n} + q_{\mathrm{ses}}/2^n\right]$.

GAME 8 *(replace* kdf *outputs with random strings in passive sessions)*: In this game we modify how keys are generated. For active flagged sessions we do as in Game 5, but for hbc sessions we want to drop kdf usage. We stress that client sessions currently tagged hbc are not completely honest: an adversary may try a replay attack utilizing the envelope from another session sid' that the server ran, while using the same $(\mathsf{uid}, s_{\mathsf{S}}^{\mathsf{uid}})$ as $\mathsf{S}^{\mathsf{sid}}$. For the current game we interpret the hbc client flag as meaning that the adversary used a (non-compromisable) honest server key and thus shouldn't be able to predict the key that will be output by the client.

When responding to NewKey commands for hbc sessions, it is immediate in Game 5 that we can always assume that a client session picks its otkAKE key $k_1 \xleftarrow{r} \{0,1\}^\kappa$ first, instead of the apparent symmetry between server and client in Game 5. This follows since our aPAKE server session always waits for the confirmation message τ' from the client before (deciding on) outputting its own key K_2. In fact, we can delay the sid test to the server key generation. Consequently, client session always generates k_1 uniformly. But more importantly, by the randomness property of the kdf, instead of hbc client inputting uniform k_1 to kdf, we directly assign random elements to (K_1, τ). Given that k_1 is generated uniformly, and that this change is distinguishable only if adversary happens to query $\mathsf{kdf}(k_1, x)$ for $x \in \{0,1\}$, we can drop kdf usage on each single NewKey except with negligible probability $q_{\mathsf{kdf}}/2^\kappa$. Note that we have finally arrived in how our ideal-world handles NewKey client queries, see Figure 4.24.

Switching to server-side hbc key generation, we see that in Game 5 we output $\perp$ except when τ' matches $\tau_{\mathsf{S}} := \mathsf{kdf}(k_2, 1)$. If k_2 is not equal to k_1, then k_2 is uniformly random and no information about it is leaked to the adversary, thus he can't predict the correct τ_{S} except with probability $1/2^\kappa$ - and if the environment fails in correctly guessing τ_{S} we output $\perp$. We

conclude that, except with probability $q_{\mathsf{NewKey}}/2^\kappa$, the only way some hbc $\mathsf{S}^{\mathsf{sid}}$ outputs a key $K_2 \neq \bot$ (and in fact this is the same key K_1 that the client outputted) is when $k_2 = k_1$ and the environment simply forwarded the honest τ from the client to the server. In particular, the client ran on the correct public key for its session, meaning it was flagged $\mathsf{hbc}(\mathsf{sid}', A, B)$ for $\mathsf{sid}' = \mathsf{sid}$. See Figure 4.24.



Responding to AKE messages:

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}(A, B)$: set $k_1 \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \alpha)$ and output $\mathsf{kdf}(k_1, 0), \mathsf{kdf}(k_1, 1)$
  2. In all other cases output $(K_1, \tau) \xleftarrow{r} \{0,1\}^\kappa \times \{0,1\}^\kappa$.

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$: If τ' was generated for $\mathsf{C}^{\mathsf{sid}}$ and the client's key K_1 was set when $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}(\mathsf{sid}, A, B)$ then output $K_2 \leftarrow K_1$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$: set $k_2 \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(\mathsf{map}^{-1}(\mathsf{IC.Dec}(h^{\mathsf{uid}}_{\mathsf{S}}, e^{\mathsf{uid}}_{\mathsf{S}})), A^{\mathsf{uid}}_{\mathsf{S}}, \alpha)$ then if $\tau' = \mathsf{kdf}(k_2, 1)$ output $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$
  3. In all other cases output $K_2 \leftarrow \bot$.

Figure 4.24: OKAPE Game 8: Removing the kdf

The difference between Game 8 and Game 5 is negligible to $\mathcal{Z}$: $|\Pr[\mathsf{G8}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}} \left[\frac{q_{\mathsf{kdf}}+1}{2^\kappa}\right] + \epsilon^{\mathcal{Z}}_{\text{kdf}}$.

Game 9 *(separating random keys from the rest of the game and dropping password usage)*: Our main goal in Game 9 is to make explicit that certain client output keys are uniform and independent of the rest of the game. By doing so, we are able to drop password usage except those that can be deferred to the functionality $\mathcal{F}_{\mathsf{aPAKE}}$. This is needed since our simulator doesn't have access to $pw^{\mathsf{uid}}_{\mathsf{S}}$ until successful password compromise. We again split this game change in several steps. Game 9(a): we start embedding freshly initialized keys (through calls to $\mathsf{SIM}_{\mathsf{AKE}}$) into every $\mathsf{IC.Dec}$ query. As is done in $\mathcal{F}_{\mathsf{otkAKE}}$, each Init call contains the owner of the key. For decryption inputs possibly used by $\mathsf{S}^{\mathsf{sid}}$, i.e. (h, y) with $y = e^{\mathsf{uid}}_{\mathsf{S}}$ and h such that $\exists(A, pw)$ with $(A, h) = \mathsf{info}^{\mathsf{uid}}_{\mathsf{S}}(pw)$, we still use the identifier S as the owner;

otherwise we use the null identity to denote keys that are not used by honest parties. The latter id is needed so that we prohibit $\mathsf{SIM}_{\mathsf{AKE}}$ computing $\mathsf{ComputeKey}$ on such keys: since the public-key B (which is generated uniformly in this context) does not leak the private key b, no adversary should be able to compute the client output key. This is needed for our second part (b) below. Note that now $\mathsf{IC.Dec}$ agrees with the definition in our ideal-world simulation. This change is indistinguishable except with probability $q_{\mathsf{IC}}\left[\epsilon_{\mathsf{map}} + \frac{q_{\mathsf{IC}}}{2^n}\right]$.

In the second step, Game 9(b) introduces the rnd flag for client sessions. These will be the client sessions for which the output key is uniform and independent of the rest of the game. In particular, it shouldn't match the key output by a server or correspond to a valid query to $\mathsf{ComputeKey}$. For instance, in a client session marked $\mathsf{hbc}(\mathsf{sid}', A, B)$ in Game 8, B is an honestly generated key for S and thus, by definition, $\mathsf{ComputeKey}$ does not allow adversary to learn this key from it - the only way this key is not independent from the rest of the game is if the server outputs the same key (i.e. S is marked hbc and $\mathsf{sid}' = \mathsf{sid}$). Note that as in Game 8, the client key generation for a session now marked rnd will output an uniform (K_1, τ) independent of any server-side computation. We modify the $\mathsf{CltSession}$ computation of Game 8 by splitting its description into the same cases as in the ideal-world Figure 6.11.

1. If $(e', s') = (e_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$: then[5] either $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, in which case we still flag this session as hbc, or $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$ and the hash h generated by the client yields a new$^{(!)}$ decryption of e', making this key rnd since we know the server key will not agree because of the mismatch between passwords. Note that by our change in (a) any new$^{(!)}$ $\mathsf{IC.Dec}$ query makes it impossible for the adversary to learn the key through $\mathsf{ComputeKey}$ and since IC has no collisions (by a previous game) $e_{\mathsf{S}}^{\mathsf{uid}}$ couldn't come from two distinct new$^{(!)}$ $\mathsf{IC.Enc}$ queries.

[5]Note that in Game 8 we also mark $(e_{\mathsf{S},\mathsf{uid}}^{\mathsf{sid}'}, s_{\mathsf{S}}^{\mathsf{uid}})$ as hbc, but in fact if $\mathsf{sid}' \neq \mathsf{sid}$ then this session is rnd. If we follow the 'switch' 1, 2, 3 and 4 we see that this case will correctly only match case 4 except with negligible probability.

- On $\text{new}^{(!)}\,(h,y)$ to $\mathsf{IC.D}$: if there exists $(\mathsf{S},\mathsf{uid})$ and (A,pw) such that $y = e_{\mathsf{S}}^{\mathsf{uid}}$ and $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (A,h)$ then set $\mathsf{id} = \mathsf{S}$, else set $\mathsf{id} = \mathsf{null}$. Initialize key B via call $(\mathsf{Init},\mathsf{id},2)$ to $\mathsf{SIM}_{\mathsf{AKE}}$ and add B to PK. Set $x' \xleftarrow{r} \mathsf{map}(B)$, add (h,x',y) to T_{IC} and send back x'

- On $(\mathsf{CltSession},\mathsf{sid},\mathsf{S},pw)$ and (e',s') to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and then

  1. If $(e',s') = (e_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$ then if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{sid}}$, else go to 4.
  2. If $\exists\, x', \mathsf{uid}$ s.t. $s' = s_{\mathsf{S}}^{\mathsf{uid}}$ and e' was output by $\text{new}^{(!)}$ $\mathsf{IC.E}$ on $(h_{\mathsf{S}}^{\mathsf{uid}}, x')$ then
     (a) if record pw_S^{uid} is $\mathsf{compromised}$ and $pw = pw_S^{\mathsf{uid}}$ then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}(A_{\mathsf{S}}^{\mathsf{uid}}, \mathsf{map}^{-1}(x'))$, jump to 5
     (b) else jump to 4
  3. If $\exists\,(x',h,a,pw')$ s.t. e' was output by $\text{new}^{(!)}$ $\mathsf{IC.E}$ on (h,x') and $\langle (pw',s'),(h,a)\rangle \in \mathsf{T}_{\mathsf{H}}$ then generate A from a and:
     (a) if $pw' = pw$: $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}(A, \mathsf{map}^{-1}(x'))$ and jump to 5
     (b) else jump to 4
  4. In all other cases set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, go to 5.
  5. Send $(\mathsf{NewSession},\mathsf{sid},\mathsf{clts},\mathsf{S},1)$ to $\mathsf{SIM}_{\mathsf{AKE}}$

Responding to AKE messages:

- On $(\mathsf{Interfere},\mathsf{sid},\mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{sid}}$ then change it to rnd

- On $(\mathsf{NewKey},\mathsf{sid},\mathsf{S},\alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:

  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, τ' was generated for $\mathsf{C}^{\mathsf{sid}}$ and the client's key K_1 was set when $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then output $K_2 \leftarrow K_1$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\exists\,(pw,B)$ s.t. $\tau' = \mathsf{kdf}(k,1)$ for $k = R_{\mathsf{S}}^{\mathsf{sid}}(B,A,\alpha)$ where $(A,h) = \mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw)$ and $(h,\mathsf{map}(B),e_{\mathsf{S}}^{\mathsf{uid}}) \in \mathsf{T}_{\mathsf{IC}}$, then (a) if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$ then output $K_2 \leftarrow \mathsf{kdf}(k,0)$; (b) otherwise go to 3
  3. In all other cases output $K_2 \leftarrow \perp$

Figure 4.25: OKAPE Game 9: rnd sessions and removing password usage

2. [6] If $\exists\ x', \mathsf{uid}$ s.t. $s' = s_{\mathsf{S}}^{\mathsf{uid}}$ and e' was output by a new$^{(!)}$ $\mathsf{IC.E}$ on $(h_{\mathsf{S}}^{\mathsf{uid}}, x')$: as the adversary runs $\mathsf{IC.Enc}$ on $h_{\mathsf{S}}^{\mathsf{uid}}$ we may assume the latter was leaked, i.e. $pw_{\mathsf{S}}^{\mathsf{uid}}$ has been compromised. The decryption done by the client (using pw) will not be new$^{(!)}$ iff $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, and in this case we still mark this session as actively attacked, otherwise the adversary can't obtain the key through a ComputeKey query and once again we can flag it as rnd.

3. If $\exists\ (x', h, a, pw')$ s.t. e' was output by new$^{(!)}$ $\mathsf{IC.E}$ on (h, x') and $\langle (pw', s'), (h, a) \rangle \in \mathsf{T_H}$: then if $pw = pw'$ we will decrypt B from a **non** new$^{(!)}$, and the adversary could learn this key from a ComputeKey query because $B \notin PK$, so this session is still marked actively attacked. On the other hand, if the passwords are distinct then, as we are assuming IC has no collisions, we have a rnd client session.

4. If none of the cases above are true, then we mark the session as rnd.

The result of the above changes is in the CltSession section of Figure 4.25.

Our third substep is Game 9 (c): we note that every Interfere on an hbc client session makes it rnd because now the only sessions being marked as honest use honestly generated server side public key B. Finally, in Game 9(d) we replace the handling of server-side NewKey queries by the description in the ideal-world (see Figure 6.11). It is clear this is just a semantical change. We also remove the public keys (A, B) from hbc client tags since they are not used. This in turn lets us drop the decryption that obtains B in CltSession and to reuse the notation from our ideal-world game for the client public key A. Lastly, we can drop the hash query $(a, h) \leftarrow \mathsf{H}(pw, s')$ since they are not used anymore for the processing of CltSession. We get $|\Pr[\mathsf{G9}] - \Pr[\mathsf{G8}]| \leq q_{\mathsf{IC}} \left[\epsilon_{\mathsf{map}} + \frac{q_{\mathsf{IC}}}{2^n}\right]$.

Game 10 *(ideal-world: cleaning up* StorePwdFile*)*: This is the ideal-world game where

[6] From here on out these sessions are marked active by Game 8, except the $\mathsf{sid} \neq \mathsf{sid}'$ case we described above, thus client session keys will have no impact in server key generations. Equivalently, we can change the flag of sessions to rnd iff k_1 can't be obtained from ComputeKey.

the interaction between the OKAPE compiler and the adversary is completely replaced by $\mathcal{F}_{\mathsf{otkAKE}}$ and our simulator $\mathsf{SIM}_{\mathsf{AKE}}$. The only difference between Game 9 and Game 10 is in $\mathsf{StorePwdFile}$: we need to drop picking $s_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \{0,1\}^{\kappa}$ and setting $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ during these queries. Looking at Figure 6.11, we see that this change is negligible to $\mathcal{Z}$ because (a) $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ is only needed after password compromise, and in this case Game 5 has already generated the $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$ pair through $\mathsf{StealPwdFile}$ and (b) $s_{\mathsf{S}}^{\mathsf{uid}}$ will be generated through either a $\mathsf{SvrSession}$ or $\mathsf{StealPwdFile}$ and it is not used before except with negligible probability. In fact, we have $\Pr[\mathsf{G10}] - \Pr[\mathsf{G9}] \leq (q_{\mathsf{H}} + q_{\mathrm{ses}})/2^{\kappa}$.

We conclude that $|\Pr[\mathsf{G10}] - \Pr[\mathsf{G0}]|$ is a negligible quantity and thus Theorem 4.4 is proved.

□

Initialize simulator $\mathsf{SIM_{AKE}}$, empty tables $\mathsf{T_{IC}}, \mathsf{T_H}$, and lists CPK, PK.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ as fresh
- On new$^{(!)}$ (pw, s) to H:
  1. If $s \neq s_{\mathsf{S}}^{\mathsf{uid}}$ for all $(\mathsf{S}, \mathsf{uid})$ then $h \xleftarrow{r} \{0,1\}^\kappa$, init. A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK
  2. If $s = s_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is compromised and $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$ set $(A, h) \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}})$
     (b) Else then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, initialize A via $(\mathsf{Init}, \mathsf{clts}_{\mathsf{S}}^{\mathsf{uid}}, 1)$ call to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \xleftarrow{r} \{0,1\}^\kappa$

     In both cases (a) and (b), set $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, h)$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$ and define a as its response, add A to CPK

  Add $\langle (pw, s), (h, a) \rangle$ to $\mathsf{T_H}$ and send back (h, a)
- On new$^{(!)}$ (h, x') to IC.E: Output $y \xleftarrow{r} Y \setminus \mathsf{T}_{\mathsf{IC}}^{h}.Y$, add (h, x', y) to $\mathsf{T_{IC}}$
- On new$^{(!)}$ (h, y) to IC.D: if there exists $(\mathsf{S}, \mathsf{uid})$ and (A, pw) such that $y = e_{\mathsf{S}}^{\mathsf{uid}}$ and $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (A, h)$ then set $\mathsf{id} = \mathsf{S}$, else set $\mathsf{id} = \mathsf{null}$. Initialize key B via call $(\mathsf{Init}, \mathsf{id}, 2)$ to $\mathsf{SIM_{AKE}}$ and add B to PK. Set $x' \xleftarrow{r} \mathsf{map}(B)$, add (h, x', y) to $\mathsf{T_{IC}}$ and send back x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: if $s_{\mathsf{S}}^{\mathsf{uid}}$ is not set, pick $s_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \{0,1\}^\kappa$. If there is a fresh $pw_{\mathsf{S}}^{\mathsf{uid}}$, mark it compromised and continue, otherwise abort. Then (a) If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$ then set $(A, h) \leftarrow \mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$; (b) else initialize A via $(\mathsf{Init}, \mathsf{clts}, 1)$ call to $\mathsf{SIM_{AKE}}$, add A to PK, pick $h \xleftarrow{r} \{0,1\}^\kappa$.

  In either case, set $(A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, h)$, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, h_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: if $s_{\mathsf{S}}^{\mathsf{uid}}$ is not set, pick $s_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \{0,1\}^\kappa$. Initialize function $R_{\mathsf{S}}^{\mathsf{sid}}$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, set $e_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} Y$, output $(e_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{clts}, 2)$ to $\mathsf{SIM_{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and (e', s') to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and:
  1. If $\exists$ uid s.t. $(e', s') = (e_{\mathsf{S}}^{\mathsf{uid}}, s_{\mathsf{S}}^{\mathsf{uid}})$:if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, else go to 4.
  2. If $\exists$ x', uid s.t. $s' = s_{\mathsf{S}}^{\mathsf{uid}}$ and e' was output by new$^{(!)}$ IC.E on $(h_{\mathsf{S}}^{\mathsf{uid}}, x')$ then
     (a) if record pw_{S}^{uid} is compromised and $pw = pw_{S}^{\mathsf{uid}}$ then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow (\mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, \mathsf{map}^{-1}(x'))$, jump to 5.
     (b) else jump to 4.
  3. If $\exists$ (x', h, a, pw') s.t. e' was output by new$^{(!)}$ IC.E on (h, x') and $\langle (pw', s'), (h, a) \rangle \in \mathsf{T_H}$ (SIM aborts if tuple not unique) then generate public key A from a and:
     (a) if $pw' = pw$: $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow (\mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}, A, \mathsf{map}^{-1}(x'))$ and jump to 5.
     (b) else jump to 4.
  4. In all other cases set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, go to 5.
  5. Send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{clts}, \mathsf{S}, 1)$ to $\mathsf{SIM_{AKE}}$

Responding to AKE messages from $\mathsf{SIM_{AKE}}$:

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = (\mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}, A, B)$ then set $k_1 \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, \alpha)$ and output $(K_1, \tau) \leftarrow (\mathsf{kdf}(k_1, 0), \mathsf{kdf}(k_1, 1))$
  2. In all other cases output $(K_1, \tau) \xleftarrow{r} \{0,1\}^\kappa \times \{0,1\}^\kappa$.
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, τ' was generated for $\mathsf{C}^{\mathsf{sid}}$ and the client's key K_1 was set when $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then output $K_2 \leftarrow K_1$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\exists$ (pw, B) s.t. $\tau' = \mathsf{kdf}(k, 1)$ for $k = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, \alpha)$ where $(A, h) = \mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw)$ and $(h, \mathsf{map}(B), e_{\mathsf{S}}^{\mathsf{uid}}) \in \mathsf{T_{IC}}$, then (a) if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$ then output $K_2 \leftarrow \mathsf{kdf}(k, 0)$; (b) otherwise go to 3
  3. In all other cases output $K_2 \leftarrow \perp$
- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: output $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ if $pk' \notin (PK \setminus CPK)$

Figure 4.26: OKAPE Game 10: $\mathcal{Z}$'s view of ideal-world interaction

Chapter 5

Randomized Half-Ideal Cipher on Groups with application to UC (a)PAKE

5.1 Introduction

The Ideal Cipher Model (ICM) dates back to the work of Shannon [118], and it models a block cipher as an Ideal Cipher (IC) oracle, where every key, even chosen by the attacker, defines an independent random permutation. Formally, an efficient adversary who evaluates a block cipher on any key k of its choice cannot distinguish computing the cipher on that key in the forward and backward direction from an interaction with oracles $E_k(\cdot)$ and $E_k^{-1}(\cdot)$, where $\{E_i\}$ is a family of random permutations on the cipher domain. The Ideal Cipher Model has seen a variety of applications in cryptographic analysis, e.g. [123, 111, 63, 115, 95, 57, 39, 90], e.g. the analysis of the Davies-Meyer construction of a collision-resistant hash [115, 39], of the Even-Mansour construction of a cipher from a public pseudorandom permutation [63],

or of the DESX method for key-length extension for block ciphers [95]. A series of works [60, 51, 82, 52, 55] shows that ICM is equivalent to the Random Oracle Model (ROM) [27]. Specifically, these papers show that n-round Feistel, where each round function is a Random Oracle (RO), implements IC for some n, and the result of Dai and Steinberger [55] shows that $n = 8$ is both sufficient and necessary. Other IC constructions include iterated Even-Mansour and key alternating ciphers [54, 17, 61], wide-input (public) random permutations [35, 33, 53], and domain extension mechanisms, e.g. [50, 75].

Ideal Ciphers on Groups: Applications. All the IC applications above consider IC on a domain of fixed-length bitstrings. However, there are also attractive applications of IC whose domain is a *group*. A prominent example is a Password Authenticated Key Exchange (PAKE) protocol called *Encrypted Key Exchange* (EKE), due to Bellovin and Meritt [28]. EKE is a compiler from plain key exchange (KE) whose messages are pseudorandom in some domain D, and it implements a secure PAKE if parties use an IC on domain D to password-encrypt KE messages.[1] The EKE solution to PAKE is attractive because it realizes UC PAKE given any key-private (a.k.a. anonymous) KEM [25], or KE with a mild "random message" property, at a cost which is the same as the underlying KE(M) *if* the cost of IC on KE(M) message domain(s) is negligible compared to the cost of KE(M) itself. However, instantiating EKE with e.g. Diffie-Hellman KE (DH-KE) [58] requires an IC on a group because DH-KE messages are random group elements.

Recently Gu et al. [74] and Freitas et al. [68] extended the EKE paradigm to cost-minimal compilers which create UC *asymmetric* PAKE (aPAKE), i.e. PAKE for the client-server setting where one party holds a one-way hash of the password instead of a password itself, from any key-hiding Authenticated Key Exchange (AKE). The AKE-to-aPAKE compilers of [74, 68] are similar to the "EKE" KE-to-PAKE compiler of [28] in that they also require

[1] Bellare et al. [26] showed that EKE+IC is a game-based secure PAKE, then Abdalla et al. [8] showed that EKE variant with explicit key confirmation realizes UC PAKE, and recently McQuoid et al. [109] showed that a round-minimal EKE variant realizes UC PAKE as well (however, see more on their analysis below).

IC-encryption of KE-related values, but they use IC to password-encrypt a KEM public key rather than KE protocol messages. The key-hiding AKE's exemplified in [74, 68], namely HMQV [99] and 3DH [108], are variants and generalizations of DH-KE where public keys are group elements, hence the AKE-to-aPAKE compilers of [74, 68] instantiated this way also require IC on a group.

Ideal Ciphers on Groups: Existing Constructions. The above motivates searching for efficient constructions of IC on a domain of an arbitrary group. Note first that a standard block cipher on a bitstring domain does not work. The elements of any group G can be encoded as bitstrings of some fixed length n, but unless these encodings cover almost all n-bit strings, i.e. unless $(1-|G|/2^n)$ is negligible, encrypting G elements under a password using IC on n-bit strings exposes a scheme to an offline dictionary attack, because the adversary can decrypt a ciphertext under any password candidate and test if the decrypted plaintext encodes a G element.

Black and Rogaway [38] showed an elegant black-box solution for an IC on G given an IC on n-bit strings provided that $c = (2^n/|G|)$ is a constant: To encrypt element $x \in G$ under key k, use the underlying n-bit IC in a loop, i.e. set x_0 to the n-bit encoding of x, and $x_{i+1} = \mathsf{IC.Enc}_k(x_i)$ for each $i \geq 0$, and output as the ciphertext the first x_i for $i \geq 1$ s.t. x_i encodes an element of group G. (Decryption works the same way but using $\mathsf{IC.Dec}$.) This procedure takes expected c uses of $\mathsf{IC.Enc}$, but timing measurement of either encryption or decryption leaks roughly $\log c$ bits of information on key k per each usage, because given the ciphertext one can eliminate all keys which form decryption cycles whose length does not match the length implied by the timing data.

To the best of our knowledge there are only two other types of constructions of IC on a group. First, the work of [60, 51, 82, 52, 55] shows that n-round Feistel network implements an IC for $n \geq 8$. Although not stated explicitly, these results imply a (randomized) IC

on a group, where one Feistel wire holds group elements, the xor gates on that wire are replaced by group operations, and hashes onto that wire are implemented as RO hashes onto the group. However, since $n = 8$ rounds is minimal [55], this construction incurs four RO hashes onto a group per cipher operation. Whereas there is progress regarding RO-indifferentiable hashing on Elliptic Curve (EC) groups, see e.g. [65], current implementations report an RO hash costs in the ballpark of 25% of scalar multiplication. Hence, far from being negligible, the cost of IC on group implemented in this way would roughly equal the DH-KE cost in the EKE compiler. The second construction of (randomized) IC combines any (randomized) quasi-bijective encoding of group elements as bitstrings with an IC on the resulting bitstrings [74]. However, we know of only two quasi-bijective encodings for Elliptic Curve groups, Elligator2 of Bernstein et al. [32] and Elligator^2 of Tibouchi et al. [121, 96], and both have some practical disadvantages. Elligator2 works for only some elliptic curves, and it can encode only half the group elements, which means that any application has to re-generate group elements until it finds one in the domain of Elligator2. Elligator^2 works for a larger class of curves, but its encoding procedure is non-constant time and it appears to be significantly more expensive than one RO hash onto a curve. Elligator^2 also encodes each EC element as a pair of underlying field elements, effectively doubling the size of the EC element representation.

IC Alternative: Programmable-Once Public Function. An alternative path was recently charted by McQuoid et al. [109], who showed that a 2-round Feistel, with one wire holding group elements, implements a randomized cipher on a group which has some IC-like properties, which [109] captured in a notion of Programmable Once Public Function (POPF). Moreover, they argue that POPF can replace IC in several applications, exemplifying it with an argument that EKE realizes UC PAKE if password encryption is implemented with a POPF in place of IC. This would be very attractive because if 2-round Feistel can indeed function as an IC replacement in applications like the PAKE of [28] or the aPAKE's of

[74, 68], this would form the most efficient and flexible implementation option for these protocols, because it works for any group which admits RO-indifferentiable hash, and it uses just one such hash-onto-group per cipher operation.

However, it seems difficult to use the POPF abstraction of [109] as a replacement for IC in the above applications because the POPF notion captures 2-round Feistel properties with game-based properties which appear not to address *non-malleability*. For that reason we doubt that it can be proven that UC PAKE is realized by EKE with IC replaced by POPF as defined in [109]. (See below for more details.) The fact that the POPF abstraction appears insufficient does not preclude that UC PAKE can be realized by EKE with encryption implemented as 2-round Feistel, but such argument would not be modular. Moreover, each application which uses 2-round Feistel in place of IC would require a separate non-modular proof. Alternatively, one could search for a "POPF+" abstraction, realized by a 2-round Feistel, which captures sufficient non-malleability properties to be useful as an IC replacement in PAKE applications, but in this work we chose a different route.

Our Results: Modified 2-Feistel as (Randomized) Half-Ideal Cipher. Instead of trying to work with 2-Feistel itself, we show that adding a block cipher BC to one wire in 2-Feistel makes this transformation non-malleable, and we capture the properties of this construction in the form of a UC notion we call a (Randomized) Half-Ideal Cipher (HIC). In Figure 5.1 we show a simple pictorial comparison of 2-Feistel, denoted $\mathsf{2F}$, and our modification, denoted $\mathsf{m2F}$. The modified 2-Feistel has the same efficiency and versatility as the 2-Feistel used by McQuoid et al. [109]: It works for any group with an RO-indifferentiable hash onto a group, it runs in fixed time, and it requires only one RO hash onto a group per cipher operation.

One drawback of $\mathsf{m2F}$ is that the ciphertext is longer than the plaintext by 2κ bits, where κ is a security parameter. However, that is less than any IC implementation above (including

POPF, which does not realize IC) except for Elligator2: IC results from n-round Feistel have loose security bounds, hence they need significantly longer randomness to achieve the same provable security; Elligator2 adds κ bits for general moduli, due to encoding of field elements as random bitstrings; Elligator^2 uses an additional field element, which adds at least 2κ bits, plus another κ bits for the field-onto-bits encoding; Finally, 2-Feistel requires at least 3κ bits of randomness when used in EKE [109].

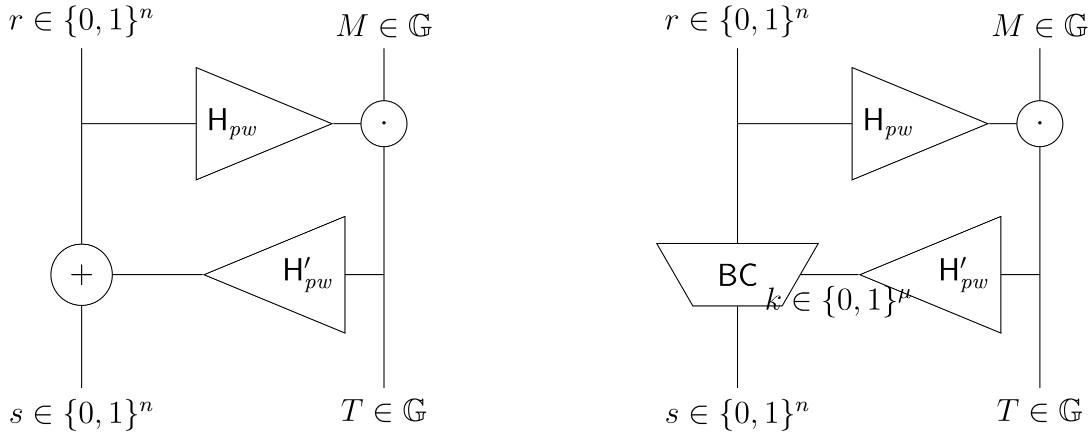


Figure 5.1: Left: two-round Feistel ($\mathsf{2F}$) used in McQuoid et al. [109]; Right: our circuit $\mathsf{m2F}$. The change from $\mathsf{2F}$ to $\mathsf{m2F}$ is small: If $k = H'(pw, T)$, then $\mathsf{2F}$ sets $s = k \oplus r$, whereas $\mathsf{m2F}$ sets $s = \mathsf{BC.Enc}(k, r)$, where BC is a block cipher.

The UC HIC notion is a relaxation of an Ideal Cipher notion, but it does not prevent applicability in protocols like [28, 74, 68], which we exemplify by showing that the following protocols remain secure with (any realization of) IC replaced by (any realization of) HIC:

(I) UC PAKE is realized by an EKE variant with IC replaced by HIC, using round-minimal KE with a random-message property;

(II) UC PAKE is realized by an EKE variant with IC replaced by HIC, using anonymous KEM with a uniform public keys property;

(III) UC aPAKE is realized by KHAPE [74] with IC replaced by HIC, using key-hiding AKE.

Regarding the first two proofs, we are not aware of full proofs exhibited for the corresponding statements where these EKE variants use IC instead of HIC, but the third proof follows the blueprint of the proof given in [74] for the KHAPE protocol using IC, and it exemplifies how little such proof changes if IC is replaced by HIC.

Half-Ideal Cipher. The first difference between IC on group $\mathbb{G}$ and HIC on group $\mathbb{G}$ is that the latter is a cipher on an extended domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$ where $\mathcal{R} = \{0,1\}^n$ is the randomness space, for $n \geq 2\kappa$ where κ is the security parameter. In the decryption direction, HIC acts exactly like IC on domain $\mathcal{D}$, i.e. unless ciphertext $c \in \mathcal{D}$ is already associated with some plaintext in the permutation table defined by key k, an adversarial decryption of c under key k returns a random plaintext m, chosen by the HIC functionality with uniform distribution over those elements in domain $\mathcal{D}$ which are not yet assigned to any ciphertext in the permutation table for key k. However, in the encryption direction HIC is only *half-ideal* in the following sense: If plaintext m is not yet associated with any ciphertext in the permutation table for key k then encryption of m under key k returns a ciphertext $c = (s, T) \in \mathcal{D} = \mathcal{R} \times \mathbb{G}$ s.t. the $T \in \mathbb{G}$ part of c *can be freely specified by the adversary*, and the $s \in \mathcal{R}$ part of c is then chosen by the HIC functionality at random with uniform distribution over s's s.t. $c = (s, T)$ is not yet assigned to any plaintext in the permutation table for key k. In short, HIC decryption on any (k, c) returns a random plaintext m (subject to the constraint that $\mathsf{RIC}(k, \cdot)$ is a permutation on $\mathcal{D}$), but HIC encryption on any (k, m) returns $c = (s, T)$ s.t. T can be correlated with other values in an arbitrary way, which is modeled by allowing the adversary to choose it, but s is random (subject to the constraint that $\mathsf{RIC}(k, \cdot)$ is a permutation).[2]

Intuitively, the reason the adversarial ability to manipulate part of IC ciphertext does not affect typical IC applications is that these applications typically rely on the following prop-

[2]This describes only the *adversarial* interface to the HIC functionality. Honest parties' interface is as in IC in both directions, except that it hides encryption randomness, i.e. encryption takes only input $M \in \mathbb{G}$ and decryption outputs only the $M \in \mathbb{G}$ part of the "extended" HIC plaintext $m \in \mathcal{D}$.

erties of IC: (1) that decryption of a ciphertext on any other key from the one used in encryption outputs a random plaintext, (2) that any change to a ciphertext implies that the corresponding plaintext is random and hence uncorrelated to the plaintext in the original ciphertext, and (3) that no two encryption operations can output the same ciphertext, regardless of the keys used, and moreover that the simulator can straight-line extract the unique key used in a ciphertext formed in the forward direction. Only properties (2) and (3) could be affected by the adversarial ability to choose the T part of a ciphertext in encryption, but the fact that the s part is still random, and that $|s| \geq 2\kappa$, means that just like in IC, except for negligible probability each encryption outputs a ciphertext which is different from all previously used ones. Consequently, just like in IC, a HIC ciphertext commits the adversary to (at most) a *single* key used to create that ciphertext in a forward direction, the simulator can straight-line extract that key, and the decryption of this ciphertext under any other key samples random elements in the domain.

Further Applications: IC domain extension, LWE-based UC PAKE. The modified 2-Feistel construction can also be used as a *domain extender* for (randomized) IC on *bitstrings*. Given an RO hash onto $\{0,1\}^t$ and an IC on $\{0,1\}^{2\kappa}$, the $\mathsf{m2F}$ construction creates a HIC on $\{0,1\}^t$, for any $t = \mathsf{poly}(\kappa)$. The modified 2-Feistelis simpler than other IC domain extenders, e.g. [50, 75], and it has better exact security bounds, hence it is an attractive alternative in applications where HIC can securely substitute for IC on a large bitstring domain. For example, by our result (II) above, $\mathsf{m2F}$ on long bitstrings can be used to implement UC PAKE from any lattice-based IND-secure and anonymous KEM. This includes several post-quantum LWE-based KEM proposals in the NIST competition, including Saber [56], Kyber [40], McEliece [13], NTRU [80], Frodo [16], and possibly others.[3] Such UC PAKE construction would add only 3κ bits in bandwidth to the underlying KEM, and its computational overhead over the underlying KEM operations would be negligible, i.e.

[3]Two recent papers [106, 124] investigate anonymity of several CCA-secure LWE-based KEMs achieved via variants of the Fujisaki-Okamoto transform [69] applied to the IND-secure versions of these KEM's. However, the underlying IND-secure KEM's are all anonymous, see e.g. [106, 124] and the references therein.

the LWE-based UC PAKE would have essentially exactly the same cost as the LWE-based unauthenticated Key Exchange, i.e. an IND-secure KEM. We show a concrete construction of UC PAKE from Saber KEM in Section 5.6.

Half-Ideal Cipher versus POPF. Our modified 2-Feistel construction and the UC HIC abstraction we use to capture its properties can be thought of as a "non-malleability upgrade" to the 2-Feistel, and to the game-based POPF abstraction used by McQuoid et al. [109] to capture its properties. One reason why the UC HIC notion is an improvement over the POPF notion is that a UC tool is easier to use in protocol applications than a game-based abstraction. More specifically, the danger of game-based properties is that they often fail to adequately capture non-malleability properties needed in protocol applications, e.g. in the EKE protocol, where the man-in-the-middle attacker can modify the ciphertexts exchanged between Alice and Bob.[4] Indeed, POPF properties seem not to capture ciphertext non-malleability. As defined in [109], POPF has two security properties, *honest simulation* and *uncontrollable outputs*. The first one says that if ciphertext c is output by a simulator on behalf of an honest party, then decrypting it under any key results in a random element in group $\mathbb{G}$, except for the (key,plaintext) pair, denoted (x^*, y^*) in [109], which was programmed into this ciphertext by the simulator. The second property says that any ciphertext c^* output by an adversary decrypts to random elements in group $\mathbb{G}$ for all keys except for key k^*, denoted x^* in [109], which was used by the adversary to create c^* in the forward direction, and which can be straight-line extracted by the simulator.[5] However, these properties do not say that the (key,plaintext) pairs behind the adversary's ciphertext c^* cannot bear any relation to the (key,plaintext) pairs behind the simulator's ciphertext c.

Note that non-malleability is necessary in a protocol application like EKE, and for that

[4] A potential benefit of a game-based notion over a UC notion is that the former *could* be easier to state and use, but this does not seem to be the case for the POPF properties of [109], because they are quite involved and subtle.

[5] Technically [109] state this property as pseudorandomness of outputs of any weak-PRF on the decryptions of c^* for any $k \neq k^*$, and not the pseudorandomness of the decrypted plaintexts themselves.

reason we think that it is unlikely that EKE can provably realize UC PAKE based on the POPF properties alone. Consider a cipher Enc on a multiplicative group s.t. there is an efficient algorithm A s.t. if $c = \mathsf{Enc}(k, M)$ and $c^* = A(c)$ then $M^* = \mathsf{Dec}(k, c^*)$ satisfies relation $M^* = M^2$ if $\mathsf{lsb}(k) = 0$, and $m^* = m^3$ if $\mathsf{lsb}(k) = 1$. If this cipher is used in EKE for password-encryption of DH-KE messages then the attacker would learn lsb of password pw used by Alice and Bob: If the attacker passes Alice's message $c_A = \mathsf{Enc}(pw, g^x)$ to Bob, but replaces Bob's message $c_B = \mathsf{Enc}(pw, g^y)$ by sending a modified message $c_B^* = A(c_B)$ to Alice, then $c_B^* = \mathsf{Enc}(pw, g^{y\cdot(2+b)})$ where $b = \mathsf{lsb}(pw)$, hence an attacker who sees Alice's output $k_A = g^{xy\cdot(2+b)}$ and Bob's output $k_B = g^{xy}$, can learn bit b by testing if $k_A = (k_B)^{(2+b)}$. More generally, any attack A which transforms ciphertext $c = \mathsf{Enc}(k, M)$ to ciphertext $c^* = \mathsf{Enc}(k^*, M^*)$ s.t. (k, M, k^*, M^*) are in some non-trivial relation, is a potential danger for EKE. We do not believe that 2-Feistel is subject to such attacks, but POPF properties defined in [109] do not seem to forbid them.

If one uses 2-Feistel directly rather than the POPF abstraction then it might still be possible to prove that EKE with 2-Feistel realizes UC PAKE. We note that 2-Feistel is subject to the following restricted form of "key-dependent malleability", which appears not to have been observed in [109] and which would have to be accounted for in such proof. Namely, consider an adversary who given ciphertext $c = (s, T)$ outputs ciphertext $c^* = (s^*, T^*)$ for any T^* and s^* s.t. $s^* \oplus \mathsf{H}'(pw^*, T^*) = s \oplus \mathsf{H}'(pw^*, T)$. Note that this adversary is not performing a decryption of c under pw^*, because it is not querying $\mathsf{H}(pw^*, r)$ for $r = s \oplus \mathsf{H}'(pw^*, T)$, but plaintexts $M^* = \mathsf{Dec}(pw, c^*)$ and $M = \mathsf{Dec}(pw, c)$ satisfy a non-trivial relation $M^*/M = T^*/T$ if $pw = pw^*$ and not otherwise. On the other hand, since this adversarial behavior seems to implement just a different form of an online attack using a unique password guess pw^*, it is still possible that EKE realizes UC PAKE even when password encryption is implemented as 2-Feistel. However, rather than considering such non-modular direct proofs for each application of IC on a group, in this paper we show that a small change in the 2-Feistel circuit implies realizing a HIC relaxation of the IC model, and this HIC relaxation

is as easy to use as IC in the security proofs for protocols like EKE [28] or aPAKE's of Gu et al. [74, 68].

Finally, we note that an extension of the above attack shows that 2-Feistel itself, without our modification, cannot realize the HIC abstraction. Observe that if the adversary computes t hashes $Z_i = \mathsf{H}(pw, r_i)$ for some pw and $r_1, ..., r_t$ and then t hashes $k_j = \mathsf{H}'(pw, T_j)$ for some $T_1, ..., T_t$, then it can combine them to form t^2 valid (plaintext, ciphertext) pairs (M_{ij}, c_{ij}) under key pw where $M_{ij} = Z_i \cdot T_j$ and $c_{ij} = (r_i \oplus k_j, T_j)$. Note that the t^2 plaintexts are formed using just $2t$ group elements $(Z_1, T_1), ..., (Z_t, T_t)$, so they are correlated. For example, the value of quotient $M_{ij}/M_{i'j}$ is the same for every j. Creating such correlations on plaintexts is impossible in the UC HIC, hence 2-Feistel by itself, without our modification, does not realize it.

5.2 Universally Composable Randomized Ideal Cipher

We define a new functionality $\mathcal{F}_{\mathsf{RIC}}$ in the UC framework ([47]), called a *Randomized (Half-)Ideal Cipher* (HIC), where the 'half' in the name refers to the fact that only half of the ciphertext is random to the adversary during encryption, as we explain below.

UC HIC is a weakening of the UC Ideal Cipher notion. Intuitively, we allow adversaries to predict or control part of the output of the cipher while the remainder is indistinguishable from random just as in the case of IC. Formally, we can interpret this as allowing the adversary to embed some tuples in the table that the functionality uses - but in a very controlled manner. We define the UC notion of Randomized Ideal Cipher via functionality $\mathcal{F}_{\mathsf{RIC}}$ in Figure 5.2.

Notes on $\mathcal{F}_{\mathsf{RIC}}$ interfaces. A randomized ideal cipherfunctionality $\mathcal{F}_{\mathsf{RIC}}$ is parametrized by domain $\mathcal{D} = \mathcal{R} \times \mathcal{G}$, where the first component is the randomness and the second is

Notation: Functionality $\mathcal{F}_{\mathsf{RIC}}$ is parametrized by domain $\mathcal{D} = \mathcal{R} \times \mathcal{G}$, and it is indexed by a session identifier sid which is a global constant, hence we omit it from notation. We denote HIC keys as passwords pw to conform to the usage of $\mathcal{F}_{\mathsf{RIC}}$ in PAKE and aPAKE applications, but keys pw are arbitrary bitstrings.

Initialization: For all $pw \in \{0,1\}^*$, initialize TRIC_{pw} as an empty table.

Interfaces for Honest Parties P*:*

on query (Enc, pw, M) from party P, for $M \in \mathcal{G}$:

sample $r \xleftarrow{\text{r}} \mathcal{R}$
if $\exists c$ s.t. $((r, M), c) \in \mathsf{TRIC}_{pw}$ then return c to P, else do:
$c \xleftarrow{\text{r}} \{\hat{c} \in \mathcal{D} : \nexists m \text{ s.t. } (m, \hat{c}) \in \mathsf{TRIC}_{pw}\}$
add $((r, M), c)$ to TRIC_{pw} and return c to P

on query (Dec, pw, c) from party P, for $c \in \mathcal{D}$:

query $(r, M) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw, c)$ and return M to P

Interfaces for Adversary $\mathcal{A}$ (or corrupt parties):

on query $(\mathsf{AdvEnc}, pw, (r, M), T)$ from adversary $\mathcal{A}$, for $(r, M) \in \mathcal{D}$ and $T \in \mathcal{G}$:

if $\exists c$ s.t. $((r, M), c) \in \mathsf{TRIC}_{pw}$ then return c to $\mathcal{A}$, else do:
if $\forall \hat{s} \in \mathcal{R}\ \exists \hat{m}$ s.t. $(\hat{m}, (\hat{s}, T)) \in \mathsf{TRIC}_{pw}\}$ then output $\perp$, else do:
$s \xleftarrow{\text{r}} \{\hat{s} \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (\hat{s}, T)) \in \mathsf{TRIC}_{pw}\}$
set $c \leftarrow (s, T)$, add $((r, M), c)$ to TRIC_{pw}, and return c to $\mathcal{A}$

on query (AdvDec, pw, c) from adversary $\mathcal{A}$, for $c \in \mathcal{D}$:

if $\exists m$ s.t. $(m, c) \in \mathsf{TRIC}_{pw}$ then return m to $\mathcal{A}$, else do:
$m \xleftarrow{\text{r}} \{\hat{m} \in \mathcal{D} : \nexists \hat{c} \text{ s.t. } (\hat{m}, \hat{c}) \in \mathsf{TRIC}_{pw}\}$
add (m, c) to TRIC_{pw} and return m to $\mathcal{A}$

Figure 5.2: Ideal functionality $\mathcal{F}_{\mathsf{RIC}}$ for *(Randomized) Half-Ideal Cipher* on $\mathcal{D} = \mathcal{R} \times \mathcal{G}$

the plaintext. Figure 5.2 separates between $\mathcal{F}_{\mathsf{RIC}}$ interfaces Enc and Dec which are used by honest parties, and the adversarial interfaces AdvEnc and AdvDec. Interfaces Enc and Dec model honest-party's usage of HIC, and they reflect our target realization of these procedures via a *randomized cipher*, i.e. a family of functions Π_{pw} s.t. for each $pw \in \{0,1\}^*$, Π_{pw} is a permutation on $\mathcal{D}$, and both Π_{pw} and Π_{pw}^{-1} are efficiently evaluable given pw. Given such cipher, algorithms $\mathsf{Enc}, \mathsf{Dec}$ can be implemented as follows: $\mathsf{Enc}(pw, M)$ picks $r \xleftarrow{\text{r}} \mathcal{R}$ and outputs $c \leftarrow \Pi_{pw}(m)$ for $m = (r, M)$, and $\mathsf{Dec}(pw, c)$ computes $m \leftarrow \Pi_{pw}^{-1}(c)$ and outputs M for $(r, M) = m$.

Functionality walk-through. Functionality $\mathcal{F}_{\mathsf{RIC}}$ reflects honest user's interfaces to randomized encryption: When an honest party P encrypts a message it specifies only $M \in \mathcal{G}$ and delegates the choice of randomness $r \stackrel{r}{\leftarrow} \mathcal{R}$ to the functionality. Similarly, when an honest party decrypts a ciphertext, the functionality discards the randomness r and reveals only M to the application. This implies that honest parties must use fresh randomness at each encryption and must discard it (or at least not use it) at decryption. By contrast, an adversary $\mathcal{A}$ has stronger interfaces than honest parties (for notational simplicity we assume corrupt parties interact to $\mathcal{F}_{\mathsf{RIC}}$ via $\mathcal{A}$), namely: (1) When $\mathcal{A}$ encrypts it can choose randomness r at will; (2) When $\mathcal{A}$ decrypts it learns the randomness r and does not have to discard it; (3) $\mathcal{A}$ can manipulate the (plaintext, ciphertext) table of each permutation Π_{pw} in the following way: If we denote ciphertexts as $c = (s, T) \in \mathcal{R} \times \mathcal{G}$, the adversary has no control of the s component of the ciphertext at encryption, i.e. it is random in $\mathcal{R}$ (up to the fact that the map has to remain a permutation), but the adversary can freely choose the T component. Items (1) and (2) are consequences of the fact that HIC is a *randomized* cipher, but item (3) is what makes this cipher *Half-Ideal*, because the adversary can control part of the value $c = \mathsf{Enc}(pw, m)$ during encryption, namely its $\mathcal{G}$ component.

The above relaxations of Ideal Cipher (IC) properties are imposed by the modified 2-Feistel construction, which in Section 5.3 we show realizes this model. However, this relaxation is harmless for many IC applications for the following reason: In a typical IC application the benefit of ciphertext randomness is that it hides the plaintext, and that it prevents the adversary from creating the same ciphertext as an encryption of two different plaintexts under two different keys. For both purposes randomness in the $s \in \mathcal{R}$ component of the ciphertext suffices as long as $\mathcal{R}$ is large enough to prevent ever encountering collisions.

The adversarial interfaces AdvEnc and AdvDec of $\mathcal{F}_{\mathsf{RIC}}$ reflect the above, and give more powers than the honest party's interfaces Enc and Dec. In encryption query AdvEnc, the adversary is allowed to pick its own randomness r *and* the $T \in \mathcal{G}$ part of the resulting ciphertext,

while its s part is chosen at random in $\mathcal{R}$ (subject to the constraint that the map remains a permutation). In decryption AdvDec, the adversary can decrypt any ciphertext $c = (s, T)$ and it learns the full plaintext $m = (r, M)$, but $\mathcal{F}_{\mathsf{RIC}}$ chooses the whole plaintext m at random. (This is another motivation for the monicker 'half-ideal': $\mathcal{F}_{\mathsf{RIC}}$ lets the adversary have some control over the ciphertext in encryption, but it does not let the adversary have any control over plaintexts in decryption.)

Our goal when designing $\mathcal{F}_{\mathsf{RIC}}$ was to keep all IC properties which are useful in applications while allowing for efficient concrete instantiation of $\mathcal{F}_{\mathsf{RIC}}$ for a group domain $\mathcal{G}$. Most importantly, if $|\mathcal{R}|$ is super-polynomial then ciphertext collisions in encryption can occur only with negligible probability, which is crucial in our HIC applications, because an adversarial ciphertext c still commits the adversary to a single key pw on which the adversary could have computed c as an encryption of some message of its choice. Secondly, just as in the case of an ideal cipher, the adversary cannot learn any information on encrypted plaintexts except via decryption with a correct decryption key. Indeed, even for an adversarially generated ciphertext c, a decryption of that ciphertext using any key $pw' \neq pw$, where pw is a unique key used in an encryption which outputted c, samples a random element from the plaintext domain.

5.3 Randomized Ideal Cipher Construction: Modified 2-Feistel

We show that Randomized (Half-)Ideal Cipher(RIC) on an arbitrary group is realized by a modifcation of the two-round Feistel, which was analyzed as a Programmable Once Public Functions (POPF) by McQuoid et al. [109], where the xor operation in the second Feistel round is replaced by an ideal block cipher BC on bitstrings. We call this construction a

modified 2-Feistel, denoted $\mathsf{m2F}$. For any group $\mathbb{G}$, construction $\mathsf{m2F}$ creates a HIC over domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$ for $\mathcal{R} = \{0,1\}^n$, using the following building blocks:

1. an ideal cipher BC on bitstring domain $\{0,1\}^n$ and key space $\{0,1\}^\mu$,
2. a random oracle hash H' with range $\{0,1\}^\mu$,
3. a random oracle hash H whose range is group $\mathbb{G}$

In essence, $\mathsf{m2F}$ creates a randomized ideal cipheron group $\mathbb{G}$ using a random oracle hash onto group $\mathbb{G}$ and an ideal cipher with n-bit blocks and μ-bit keys. The exact security analysis shows that it suffices if n and μ are both set to 2κ.

For each key pw, function $\mathsf{m2F}_{pw}$ is pictorially shown in Figure 5.1. Here we define it by the algorithms which compute $\mathsf{m2F}_{pw}$ and $\mathsf{m2F}_{pw}^{-1}$. (Throughout the paper we denote group $\mathbb{G}$ operation as a multiplication, but this is purely a notational choice, and the construction applies to additive groups as well.)

$$\mathsf{m2F}_{pw} \;:\; \{0,1\}^n \times \mathbb{G} \;\;\rightarrow\;\; \{0,1\}^n \times \mathbb{G} \tag{5.1}$$

where:

$\mathsf{m2F}_{pw}(r, M)$:

1. $T \leftarrow M/\mathsf{H}(pw, r)$
2. $k \leftarrow \mathsf{H}'(pw, T)$
3. $s \leftarrow \mathsf{BC.Enc}(k, r)$
4. Output (s, T)

$\mathsf{m2F}_{pw}^{-1}(s, T)$:

1. $k \leftarrow \mathsf{H}'(pw, T)$
2. $r \leftarrow \mathsf{BC.Dec}(k, s)$
3. $M \leftarrow \mathsf{H}(pw, r) \cdot T$
4. Output (r, M)

The following theorem captures the security of the $\mathsf{m2F}$ construction:

Theorem 5.1. *Construction* $\mathsf{m2F}$ *realizes functionality* $\mathcal{F}_{\mathsf{RIC}}$ *in the domain* $\mathcal{R} \times \mathbb{G}$ *for* $\mathcal{R} = \{0,1\}^n$ *if* $\mathsf{H} : \{0,1\}^* \times \{0,1\}^n \to \mathbb{G}$, $\mathsf{H}' : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\mu$ *are random oracles,* $\mathsf{BC} : \{0,1\}^\mu \times \{0,1\}^n \to \{0,1\}^n$ *is an ideal cipher, and* μ *and* n *are both* $\Omega(\kappa)$ *where* κ *is a security parameter.*

Proof. The proof for Theorem 5.1 must exhibit a simulator algorithm SIM, which plays a role of an ideal-world adversary interacting with functionality $\mathcal{F}_{\mathsf{RIC}}$, and show that no efficient environment $\mathcal{Z}$ can distinguish, except for negligible probability, between (1) a *real-world game*, i.e. an interaction with (1a) honest parties who execute $\mathcal{Z}$'s encryption and decryption queries using Enc and Dec implemented with circuit $\mathsf{m2F}$, and (1b) RO/IC oracles H, H', BC, BC^{-1}, and (2) an *ideal-world game*, i.e. an interaction with (2a) parties P who execute $\mathcal{Z}$'s encryption and decryption queries using interfaces $\mathsf{Enc}, \mathsf{Dec}$ of $\mathcal{F}_{\mathsf{RIC}}$, and (2b) simulator SIM, who services $\mathcal{Z}$'s calls to H, H', BC, BC^{-1} using interfaces AdvEnc and AdvDec of $\mathcal{F}_{\mathsf{RIC}}$.

We start by describing the simulator algorithm SIM, shown in Figure 5.3. Note that SIM interacts with an adversarial environment algorithm $\mathcal{Z}$ by servicing $\mathcal{Z}$'s queries to the RO and IC oracles $\mathsf{H}, \mathsf{H}', \mathsf{BC}, \mathsf{BC}^{-1}$. Intuitively, SIM populates input, output tables for these functions, $\mathsf{TH}, \mathsf{TH}'$ and TBC, in the same way as these idealized oracles would, except when SIM detects a possible encryption or decryption computation of the modified 2-Feistelcircuit. In case SIM decides that these queries form either computation of $\mathsf{m2F}$ or $\mathsf{m2F}^{-1}$ on new input, SIM detects that input, invokes the adversarial interfaces AdvEnc or AdvDec of $\mathcal{F}_{\mathsf{RIC}}$ to find the corresponding output, and it embeds proper values into these tables to emulate the circuit leading to the computation of this output. The detection of $\mathsf{m2F}$ and $\mathsf{m2F}^{-1}$ evaluation is relatively straightforward: First, SIM treats every $\mathsf{BC.Dec}$ query (k, s) as a possible $\mathsf{m2F}^{-1}$ evaluation on key pw and ciphertext $c = (s, T)$ for T s.t. $k = \mathsf{H}'(pw, T)$. If it is, SIM queries $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ on (pw, c) to get $m = (r, M)$. Since this is a random sample from the HIC domain, with overwhelming probability H was not queried on r so SIM can set

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on adversary's query $\mathsf{H}(pw, r)$

if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$:
  $h \xleftarrow{r} \mathbb{G}$
  add (pw, r, h) to TH
return h

on adversary's query $\mathsf{H}'(pw, T)$

if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
  $k \xleftarrow{r} \{0,1\}^\mu$
  abort if $\exists(\hat{pw}, \hat{T})$ s.t. $(\hat{pw}, \hat{T}, k) \in \mathsf{TH}'$ *(kcol.abort)*
  abort if $\exists(\hat{r}, \hat{s})$ s.t. $(k, \hat{r}, \hat{s}) \in \mathsf{TBC}$ *(bckey.abort)*
  add (pw, T, k) to TH'
return k

on adversary's query $\mathsf{BC.Enc}(k, r)$

if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw, T)$[a]:
    $M \leftarrow \mathsf{H}(pw, r) \cdot T$
    $(s, \hat{T}) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$
    abort if $\hat{T} \neq T$ or AdvEnc outputs $\perp$ *(advenc.abort)*
  else:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k, \hat{r}, s) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return s

[a] If it exists, we denote by $\mathsf{TH}'(pw, T)$ the (unique) k s.t. $(pw, T, k) \in \mathsf{TH}'$

on adversary's query $\mathsf{BC.Dec}(k, s)$

if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw, T)$:
    $(r, M) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw, (s, T))$
    abort if $\exists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}$ *(advdec.abort)*
    abort if $\exists h$ s.t. $(pw, r, h) \in \mathsf{TH}$ *(rcol.abort)*
    add $(pw, r, M \cdot T^{-1})$ to TH
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return r

Figure 5.3: Simulator SIM for the proof of Theorem 5.1

$\mathsf{H}(pw, r)$ to M/T. Second, SIM treats every $\mathsf{BC.Enc}$ query (k, r) as possible $\mathsf{m2F}$ evaluation on (r, M) s.t. $M = \mathsf{H}(pw, r) \cdot T$ for T s.t. $k = \mathsf{H}'(pw, T)$. However, here is where the difference between IC and HIC shows up: The $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ query fixes the encryption of $m = (r, M)$ to $c = (s, T)$, and whereas s can be random (and SIM can set $\mathsf{BC.Enc}(k, r) := s$ for any $c = (s, T)$ returned by $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ as encryption of m under key pw), value T was fixed by H' output k (except for the negligible probability of finding collisions in H'). This is why our $\mathcal{F}_{\mathsf{RIC}}$ model allows the simulator, i.e. the ideal-world adversary, to fix the T part of the ciphertext in the adversarial encryption query AdvEnc.

Proof Overview. The proof must show that for any environment $\mathcal{Z}$, its view of the real-world game defined by algorithms $\mathsf{Enc}, \mathsf{Dec}$ which use the randomized cipher $\mathsf{m2F}$, and the

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

For each $pw \in \{0,1\}^*$, initialize empty sets TRIC_{pw} and usedR_{pw}.

define $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw,(r,M),T)$:

if $\nexists\, c$ s.t. $((r,M),c) \in \mathsf{TRIC}_{pw}$:
  $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : (*,(\hat{s},T)) \notin \mathsf{TRIC}_{pw}\}$
  $c \leftarrow (s,T)$
  add $((r,M),c)$ to TRIC_{pw}
return c

define $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw,(s,T))$:

if $\nexists\, (r,M)$ s.t. $((r,M),(s,T)) \in \mathsf{TRIC}_{pw}$:
  $(r,M) \xleftarrow{r} \mathcal{D}$
  abort if $\exists\, \hat{c}$ s.t. $((r,M),\hat{c}) \in \mathsf{TRIC}_{pw}$
  abort if $r \in \mathsf{usedR}_{pw}$ else add r with tag $\mathsf{m2F}$
  add $((r,M),(s,T))$ to TRIC_{pw}
return M

on query $\mathsf{Enc}(pw,M)$:

$r \xleftarrow{r} \{0,1\}^n$
abort if $r \in \mathsf{usedR}_{pw}$, else add r with tag $\mathsf{m2F}$
if $\nexists\, c$ s.t. $((r,M),c) \in \mathsf{TRIC}_{pw}$:
  $c \xleftarrow{r} \{\hat{c} : \nexists\, \hat{m}$ s.t. $(\hat{m},\hat{c}) \in \mathsf{TRIC}_{pw}\}$
  add $((r,M),c)$ to TRIC_{pw}
return c

on query $\mathsf{Dec}(pw,c)$:

$(r,M) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw,c)$
return M

on query $\mathsf{H}(pw,r)$

abort if $r \in \mathsf{usedR}_{pw}$ tagged $\mathsf{m2F}$, else add r
if $\nexists\, h$ s.t. $(pw,r,h) \in \mathsf{TH}$:
  $h \xleftarrow{r} \mathbb{G}$
  add (pw,r,h) to TH
return h

on query $\mathsf{H}'(pw,T)$

if $\nexists\, k$ s.t. $(pw,T,k) \in \mathsf{TH}'$:
  $k \xleftarrow{r} \{0,1\}^\mu$
  abort if $\exists\, (\hat{pw},\hat{T})$ s.t. $(\hat{pw},\hat{T},k) \in \mathsf{TH}'$
  abort if $\exists\, (\hat{r},\hat{s})$ s.t. $(k,\hat{r},\hat{s}) \in \mathsf{TBC}$
  add (pw,T,k) to TH'
return k

on query $\mathsf{BC.Enc}(k,r)$

if $k = \mathsf{TH}'(pw,T)$:
  abort if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$
  else add r to usedR_{pw}
if $\nexists\, s$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $M \leftarrow \mathsf{H}(pw,r) \cdot T$
    $(s,\hat{T}) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw,(r,M),T)$
    abort if $\hat{T} \neq T$
  else:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists\, \hat{r}$ s.t. $(k,\hat{r},s) \in \mathsf{TBC}\}$
  add (k,r,s) to TBC
return s

on query $\mathsf{BC.Dec}(k,s)$

if $\nexists\, r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $(r,M) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw,(s,T))$
    abort if $\exists\, \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}$
    abort if $\exists\, h$ s.t. $(pw,r,h) \in \mathsf{TH}$
    add $(pw,r,M \cdot T^{-1})$ to TH
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists\, \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}\}$
  add (k,r,s) to TBC
if $k = \mathsf{TH}'(pw,T)$:
  remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$
return r

Figure 5.4: The ideal-world Game 00, and its modification Game 11 (text in gray)

Game 2: replacing decryption by circuit

on query $\mathsf{m2F.Dec}(pw,(s,T))$:
$k \leftarrow \mathsf{H}'(pw,T)$
$r \leftarrow \mathsf{BC.Dec}(k,s)$
$M \leftarrow \mathsf{H}(pw,r) \cdot T$
if $\mathsf{m2F.Dec}$ query was fresh, add tag $\mathsf{m2F}$ to $r \in \mathsf{usedR}_{pw}$
return M

Game 3: Enc calls AdvDec

on query $\mathsf{m2F.Enc}(pw,M)$:
$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it with tag $\mathsf{m2F}$
if $\nexists c$ s.t. $((r,M),c) \in \mathsf{TRIC}_{pw}$:
    $T \xleftarrow{r} \mathbb{G}$
    $c \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw,(r,M),T)$
return c

Game 4: replacing encryption by circuit

on query $\mathsf{m2F.Enc}(pw,M)$:
$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort
$T \leftarrow M/\mathsf{H}(pw,r)$
$k \leftarrow \mathsf{H}'(pw,T)$
$s \leftarrow \mathsf{BC.Enc}(k,r)$
assign tag $\mathsf{m2F}$ to r in the set usedR_{pw}
return (s,T)

Game 5: H is a random oracle

$\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ not used anymore

on query $\mathsf{BC.Dec}(k,s)$:

if $\nexists\, r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
    if $k = \mathsf{TH}'(pw,T)$:
        $r \xleftarrow{r} \{0,1\}^n$
        if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
        $h \leftarrow \mathsf{H}(pw,r)$
        $M \leftarrow h \cdot T$
        if $\exists \hat{c}$ s.t. $((r,M),\hat{c}) \in \mathsf{TRIC}_{pw}$ then abort
        add $((r,M),(s,T))$ to TRIC_{pw}
    else:
        $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k,r,\hat{s}) \in \mathsf{TBC}$
    add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

Game 6: simplifying parameters

define $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw,r,T)$:
if $\nexists s$ s.t. $(r,(s,T)) \in \mathsf{TRIC}_{pw}$:
    $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (\hat{r},(\hat{s},T)) \in \mathsf{TRIC}_{pw}\}$
    add $(r,(s,T))$ to TRIC_{pw}
return s

on query $\mathsf{BC.Dec}(k,s)$:
if $\nexists r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
    if $k = \mathsf{TH}'(pw,T)$:
        $r \xleftarrow{r} \{0,1\}^n$
        if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
        query $\mathsf{H}(pw,r)$ and discard the output
        if $\exists \hat{c}$ s.t. $(r,\hat{c}) \in \mathsf{TRIC}_{pw}$ then abort
        add $(r,(s,T))$ to TRIC_{pw}
    else:
        $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k,r,\hat{s}) \in \mathsf{TBC}$
    add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

on query $\mathsf{BC.Enc}(k,r)$:
if $k = \mathsf{TH}'(pw,T)$:
    if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$ then abort
        else add r to usedR_{pw}
if $\nexists s$ s.t. $(k,r,s) \in \mathsf{TBC}$:
    if $k = \mathsf{TH}'(pw,T)$:
        query $\mathsf{H}(pw,r)$ and discard the output
        $s \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw,r,T)$
    else:
        $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k,\hat{r},s) \in \mathsf{TBC}\}$
    add (k,r,s) to TBC
return s

Game 7: using k

Initialization: $\forall\, k$ initialize empty TRIC_k

define $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(k,r)$:
if $\nexists s$ s.t. $(r,s) \in \mathsf{TRIC}_k$:
    $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (\hat{r},\hat{s}) \in \mathsf{TRIC}_k\}$
    add (r,s) to TRIC_k
return s

on query $\mathsf{BC.Dec}(k,s)$:
if $\nexists r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
    if $k = \mathsf{TH}'(pw,T)$:
        $r \xleftarrow{r} \{0,1\}^n$
        if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
        if $\exists \hat{s}$ s.t. $(r,\hat{s}) \in \mathsf{TRIC}_k$ then abort
        add (r,s) to TRIC_k
    else:
        $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k,r,\hat{s}) \in \mathsf{TBC}$
    add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

on query $\mathsf{BC.Enc}(k,r)$:
if $k = \mathsf{TH}'(pw,T)$:
    if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$ then abort
        else add r to usedR_{pw}
if $\nexists s$ s.t. $(k,r,s) \in \mathsf{TBC}$:
    if $k = \mathsf{TH}'(pw,T)$:
        $s \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(k,r)$
    else:
        $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k,\hat{r},s) \in \mathsf{TBC}\}$
    add (k,r,s) to TBC
return s

Figure 5.5: Game-changes (part 1) in the proof of Theorem 5.1

Game 8: TRIC is redundant	

<u>Initialization</u>: Drop TRIC usage.
$\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ not used anymore
<u>on query $\mathsf{BC.Enc}(k, r)$:</u>
if $k = \mathsf{TH}'(pw, T)$:
    if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$ then abort
            else add r to usedR_{pw}
if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k, \hat{r}, s) \in \mathsf{TBC}\}$
    add (k, r, s) to TBC
return s
	<u>on query $\mathsf{BC.Dec}(k, s)$:</u>
	if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
	    if $\exists (pw, T)$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
	        $r \xleftarrow{r} \{0,1\}^n$
	        if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
	    else:
	        $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k, r, \hat{s}) \in \mathsf{TBC}\}$
	    add (k, r, s) to TBC
	remove tag $\mathsf{m2F}$ from $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw, T)$
	return r

Figure 5.6: Game-changes (part 2) in the proof of Theorem 5.1

ideal-world game defined by functionality $\mathcal{F}_{\mathsf{RIC}}$ and simulator SIM of Figure 5.3. The proof starts from the ideal-world view, which we denote as Game 00, and via a sequence of games, each of which we show is indistinguishable from the next, it reaches the real-world view, which we denote as Game 99.

We include the details of all the game changes and reductions in later paragraphs. To give a glance at our proof strategy here we provide the code of all successive games in Figures 5.4, 5.5, and 5.6. Figure 5.4 describes the ideal-world Game 00 and its mild modification Game 11. All these games, starting from Game 00 in Figure 5.4, interact with an adversarial environment $\mathcal{Z}$, and each game provides two types of interfaces corresponding two types of $\mathcal{Z}$'s queries: (a) the honest party's interfaces $\mathsf{Enc}, \mathsf{Dec}$, which $\mathcal{Z}$ can query via any honest party, and (b) RO/IC oracles $\mathsf{H}, \mathsf{H}', \mathsf{BC}, \mathsf{BC}^{-1}$, which $\mathcal{Z}$ can query via its "real-world adversary" interface. Figure 5.4 defines two sub-procedures, $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ and $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$, whose code matches exactly the corresponding interfaces of $\mathcal{F}_{\mathsf{RIC}}$. These subprocedures are used internally by Game 00: They are invoked by the code that services $\mathcal{Z}$'s queries $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$, because Game 00 follows SIM's code on these queries, and AdvDec is also invoked by Dec, because this is how $\mathcal{F}_{\mathsf{RIC}}$ implements Dec.

Figures 5.5 and 5.6 describe the modifications created by all subsequent games, except for the last one, the real-world game denoted Game 99, which is very similar to Game 88, which is the last game shown in Figure 5.6. Let $q_{\mathsf{m2F}}, q_{\mathsf{BC}}, q_{\mathsf{H}}$ be the number of environment queries

to resp. honest parties' oracles $\mathsf{Enc}, \mathsf{Dec}$, block cipher $\mathsf{BC}, \mathsf{BC}^{-1}$ oracles (denoted $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ in Figs. 5.4-5.6), and random oracles H/H', and let $q = (q_{\mathsf{m2F}} + q_{\mathsf{BC}} + q_{\mathsf{H}})$.

Below we present the game changes used in our proof of Theorem 5.1. We refer the reader to Section 5.3 for the notation, and to Figures 5.4 and 5.5 for the specification of all successive games.

Let P_i be the probability that the environment $\mathcal{Z}$ outputs 1 when interacting with the i-th game. We will show that $|P_i - P_{i+1}|$ is negligible for every i.

Game 0 *(ideal world)*: This is the ideal-world game played between $\mathcal{Z}$ and SIM. We describe it formally in Figure 5.4, except that Game 0 omits the gray boxes and AdvDec is answered just as in our $\mathcal{F}_{\mathsf{RIC}}$ construction: $(r, M) \xleftarrow{r} \{m \in \{0,1\}^n \times \mathbb{G} : \nexists \hat{c} \text{ s.t. } (\hat{m}, \hat{c}) \in \mathsf{TRIC}_{pw}\}$.

Game 1 *(adding* usedR *and randomizing* AdvDec*)*: In this first game change we start by randomizing AdvDec and then adding aborts for certain accesses to the randomness r used by $\mathsf{m2F}$, see Figure 5.4. We randomize $\mathsf{AdvDec}(pw, (s, T))$ by picking $(r, M) \xleftarrow{r} \mathcal{D}$ and then aborting if this pair $((r, M), (s, T))$ happens to be in the table TRIC_{pw}. This is clearly a negligible change, in fact, as long as this negligible abort (which happens with probability at most $|\mathsf{TRIC}_{pw}|/(2^n \cdot |\mathbb{G}|)$) does not happen then the games are the same.

Moreover, we want to avoid distinct (fresh)[6] calls to Enc and Dec reusing the same r (or them being called after $\mathsf{TH}(pw, r)$ has been set). This is motivated by the fact that the Feistel circuit would impose, for a plaintext-ciphertext pair $((r, M), (s, T)) \in \mathsf{TRIC}_{pw}$, the relation $M/T = \mathsf{H}(pw, r)$. If the same (pw, r) pair were used for multiple calls, then we can't expect M/T to be the same except with negligible probability, and thus we wouldn't be able to embed the correct value (since there are multiple) into TH. Similarly, if the r used by $\mathsf{m2F}$ is already at TH, the adversary could notice the discrepancy to the relation in the

[6]In this paragraph, and henceforth, a call usually, but not always, implicitly means a fresh call, i.e., this call is not a simple table lookup.

Feistel circuit. In fact, in the current game a direct call to $\mathsf{m2F}$ by the environment has no relationship to the other oracle tables, and in particular we need to disallow the adversary to query $\mathsf{TH}(pw, r)$ right after such a $\mathsf{m2F}$ invocation. This is a valid game change (i.e. the adversary can't force such an abort except with negligible probability) since the r used is not leaked, neither for a Dec nor an Enc call. The exception is when $\mathcal{Z}$ does run the decryption circuit (hence learning r through the $\mathsf{BC.Dec}$ call that is part of it) after the $\mathsf{m2F}$ call. To avoid this we introduce a flag denoted $\mathsf{m2F}$ for such calls before the decryption circuit is attempted.

Now it is clear that the size $\sum_r |\mathsf{usedR}_{pw}|$ of the set of used r is bound by $q_{\mathsf{m2F}} + q_{\mathsf{BC}}$. Hence no usedR abort happens except with (negligible) probability $(q_{\mathsf{m2F}}+q_{\mathsf{BC}})\cdot(q_{\mathsf{m2F}}+q_{\mathsf{BC}}+q_{\mathsf{H}})/2^n$.

Therefore this game change is indistinguishable except with probability

$$|P_0 - P_1| \leq \frac{(q_{\mathsf{m2F}} + q_{\mathsf{BC}})^2}{2^n \cdot |\mathbb{G}|} + \frac{(q_{\mathsf{m2F}} + q_{\mathsf{BC}}) \cdot (q_{\mathsf{m2F}} + q_{\mathsf{BC}} + q_{\mathsf{H}})}{2^n} \leq 2 \cdot \frac{q^2}{2^n} \tag{5.2}$$

GAME 2 *(replacing decryption* $\mathsf{m2F.Dec}$ *by circuit)*: We replace queries to $\mathsf{m2F.Dec}$ by the circuit of our construction. First we argue that this is a valid change for a fresh $\mathsf{m2F.Dec}$ query, see Fig. 5.7.

In this simplified case the $\mathsf{BC.Dec}$ call is also fresh, in either the current game or the previous. This implies that it calls a fresh AdvDec itself, fixing the H table so that the output M is the same as in Game 1, namely it comes from an AdvDec query. Suppose instead that $(k, r, s) \in \mathsf{TBC}$ for some r. Then this triple was added to the table by either a $\mathsf{BC.Enc}$ or $\mathsf{BC.Dec}$ query. The latter cannot happen since such a query would have inputs (k, s) and therefore its $\mathsf{AdvDec}(pw, (s, T))$ call would have populated TRIC_{pw} - note that we are using the fact that bckey.col abort was not reached. Similarly, a $\mathsf{BC.Enc}(k, r)$ that returns s would

have run $(s, T) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$ so that the $\mathsf{m2F.Dec}$ query couldn't be fresh either. We conclude that the newly introduced $\mathsf{BC.Dec}$ calls are fresh, and that these queries make fresh calls to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$.

But then the internal AdvDec call is generating (r, M) uniformly, so r is uniform. There are only two side effect of this game change in the environment's view: 1) $h \leftarrow \mathsf{TH}(pw, r)$ now satisfies $h = M/T$, while in the previous game this is not true right after the $\mathsf{m2F.Dec}$ query (an $\mathsf{H}(pw, r)$ query would return an independent, uniform value) and 2) (k, r, s) is added to TBC where $k = \mathsf{H}'(pw, T)$. But we added r to usedR with flag $\mathsf{m2F}$ and our usedR aborts in Game 1 guarantee that there is no call $\mathsf{H}(pw, r)$ or $\mathsf{TBC}(k, r)$ before the adversary itself runs the decryption circuit, i.e. $\mathsf{BC.Dec}(k, s)$. But this call would embed the same relationship in the TH and TBC table since bckey.abort does not happen, so this change is not visible to the adversary.

We do need to take into consideration the new aborts that are possible by this game change. The H' query that is now implicitly called by $\mathsf{m2F.Dec}$ can only abort if it is fresh and there is a collision with the H' table or TBC table (see definition of H' queries in Figure 5.3). This happens with probability at most $(|\mathsf{TH}'| + |\mathsf{TBC}|)/2^{\mu}$ for each added H' call. The $\mathsf{BC.Dec}$ procedure, which if fresh is executing its innermost if, will only abort if either AdvDec aborts, advdec.abort is reached or rcol.abort happens. As we argued in the previous paragraph, r is uniform hence we obtain the negligible probability bound $|\mathsf{TRIC}_{pw}|/(2^n \times |\mathbb{G}|) + |\mathsf{usedR}_{pw}|/2^n + |\mathsf{TBC}_k|/2^n + |\mathsf{TH}_{pw}|/2^n \leq 4 \cdot q/2^n$.

Finally, looking at our argument above, we see that even if $\mathsf{BC.Dec}(k, s)$ was not a fresh query during a (necessarily non fresh) $\mathsf{m2F.Dec}$ call, the r paired with (k, s) in the table TBC satisfies $(r, \mathsf{TH}(pw, r) \cdot T) = \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw, (s, T))$ where $k = \mathsf{TH}'(pw, T)$. As we saw above this follows from how both $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ are defined. We conclude that the

change to the circuit is valid even for non fresh AdvDec queries and we get

$$|P_2 - P_1| \leq q^2 \cdot \left\{ \frac{2}{2^\mu} + \frac{4}{2^n} \right\} \tag{5.3}$$

on query $\mathsf{m2F.Dec}(pw, (s, T))$:
if $\nexists (r, M)$ s.t.
$((r, M), (s, T)) \in \mathsf{TRIC}_{pw}$:
  $k \leftarrow \mathsf{H}'(pw, T)$
  $r \leftarrow \mathsf{BC.Dec}(k, s)$
  $M \leftarrow \mathsf{H}(pw, r) \cdot T$
  add tag $\mathsf{m2F}$ to $r \in \mathsf{usedR}_{pw}$
return M

Figure 5.7: Fresh queries to $\mathsf{m2F.Dec}$ are replaced by the circuit

Game 3 *(using* AdvEnc *to answer* Enc *queries)*: We replace $\mathsf{m2F.Enc}(pw, M)$ by a call to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$ using uniform T. The goal is to link the AdvEnc queries done in $\mathsf{BC.Enc}$ with the way $\mathsf{m2F.Enc}$ is computed - this will help with our next goal of changing Enc to match the circuit. This modification skews the distribution of TRIC_{pw}, but the statistical difference this introduces is negligible. The difference is that in Game 2 (s, T) is chosen uniformly from set $\{c \in \{0,1\}^n \times \mathbb{G} : \nexists \hat{m} \text{ s.t. } (\hat{m}, c) \in \mathsf{TRIC}_{pw}\}$, while in Game 3 first T is chosen uniformly in $\mathbb{G}$ and then s is chosen uniformly from set $\{s \in \{0,1\}^n : \nexists \hat{m} \text{ s.t. } (\hat{m}, (s, T)) \in \mathsf{TRIC}_{pw}\}$. Since there are at most q elements in table TRIC_{pw}, the skew this introduces on the distribution of a chosen pair (s, T) is at most $4q/(2^n \cdot |\mathbb{G}|)$ per encryption query, leading to the following upper bound:

$$|P_3 - P_2| \leq \frac{4 \cdot q \cdot q_{\mathsf{Enc}}}{2^n \cdot |\mathbb{G}|} \leq 4 \cdot \frac{q^2}{2^n \cdot |\mathbb{G}|} \tag{5.4}$$

Game 4 *(*$\mathsf{m2F.Enc}$ *can also be replaced by the two-round Feistel circuit)*: We now assert that replacing Enc from the previous game by the $\mathsf{m2F}$ encryption circuit is also a valid game

change.

Since we check $r \notin \mathsf{usedR}_{pw}$, the r that is picked by Enc does not appear in the H table and thus $T \leftarrow M/\mathsf{H}(pw, r)$ will assign an uniform value to T just as Game 3 does. The $s \leftarrow \mathsf{BC.Enc}(k, r)$ call, much like our $\mathsf{BC.Dec}$ query in Game 2, will in turn call AdvEnc indirectly for $\mathsf{m2F.Enc}$ making the output of the latter the same as in the previous game. As in Game 2, the side effect of modifying Enc in this way is that now we have certain relationships between the table values. But since r is by definition not leaked by Enc this is a negligible change just as before. In fact, looking at the newly introduced aborts, we see that the only possible ones (note we may assume $r \notin \mathsf{usedR}_{pw}$) are the ones inside the H' query. This leads us to the bound

$$|P_4 - P_3| \leq \frac{2q^2}{2^\mu} \tag{5.5}$$

Game 5 *(H is a random oracle)*: If we are to reach the real-world game described in Figure 9, we need to show that H is indistinguishable from a random oracle. Currently, the only obstacle in the way of this proof is that TH is not only modified in response to a (direct or indirect) H query, but it is also changed during a $\mathsf{BC.Dec}$ call. In this game we drop AdvDec usage in $\mathsf{BC.Dec}$ and make clearer that this modification to TH is still uniform. We start this process by expanding $\mathsf{BC.Dec}$, see Figure 5.8.

In fact, if we look thoroughly at the current game, we notice that the innermost else in this figure of the expanded $\mathsf{BC.Dec}$ query will never be reached. Namely, say a (necessarily fresh) $\mathsf{BC.Dec}$ query reaches this line in the execution. Then there is m such that $(m, (s, T))$ is in TRIC_{pw}. But since we removed direct AdvEnc and AdvDec queries from $\mathsf{m2F}$ invocations, this tuple $((r, M), (s, T))$ must have been added to TRIC_{pw} by a BC query. The only $\mathsf{BC.Dec}$

```
on query BC.Dec(k, s)

if ∄ r s.t. (k, r, s) ∈ TBC:
    if k = TH'(pw, T):
        if ∄m s.t. (m, (s, T)) ∈ TRIC_pw:
            (r, h) <-r- D
            if r ∈ usedR_pw abort, else add r to it with tag m2F
            M <- h · T
            if ∃ĉ s.t. ((r, M), ĉ) ∈ TRIC_pw then abort
            add ((r, M), (s, T)) to TRIC_pw
        else:
            let ((r, M), (s, T)) ∈ TRIC_pw
            h <- M · T^-1
        if ∃ŝ s.t. (k, r, ŝ) ∈ TBC then abort                (advdec.abort)
        if ∃ĥ s.t. (pw, r, ĥ) ∈ TH then abort                  (rcol.abort)
        add (pw, r, h) to TH
    else:
        r <-r- {r ∈ {0,1}^n : ∄ŝ s.t. (k, r, ŝ) ∈ TBC}
    add (k, r, s) to TBC
remove tag m2F from record r ∈ usedR_pw if k = TH'(pw, T)
return r
```

Figure 5.8: Expanding BC.Dec

query that could have caused $((\cdot,\cdot),(s,T))$ to have been added to TRIC_{pw} is one with (k,s) as input, which would make the current query not fresh (i.e. a contradiction). Similarly, a BC.Enc query couldn't have added $((\cdot,\cdot),(s,T))$ to TRIC_{pw} since this implies that $(k,\cdot,s)$ would have been added to TBC - which again would contradict the freshness of the current BC.Dec query.

Moreover, considering the above we can conclude that if either $\exists \hat{h}$ s.t. $(pw,r,\hat{h}) \in \mathsf{TH}$ or $\exists \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}$ with $k = \mathsf{TH}'(pw,T)$ is true, then $r \in \mathsf{usedR}_{pw}$. In particular, if the latter is not the case then a call to $\mathsf{H}(pw,r)$ returns an uniform h. So we can let a H query in BC.Dec pick h by itself, instead of doing $(r,h) \xleftarrow{r} \mathcal{D}$ ourselves. We can also assume that (k,r) is available in the TBC table, so that we are allowed to drop this abort in BC.Dec as it is already caught by the usedR_{pw} abort.

The above remarks allows us to simplify BC.Dec considerably for Game 5 while not changing the view of the environment: $P_5 = P_4$.

GAME 6 *(simplifying parameters)*: With our previous game changes TRIC_{pw} is now only

accessed/modified by (possibly indirect) $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ queries. It is clear from their definition that any call to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ uses the correct T, namely these calls return $\hat{T} = T$. So there is no need to return $\hat{T}$ and we can drop this component of the output. Similarly, any tuple $((r, M), (s, T)) \in \mathsf{TRIC}_{pw}$ satisfies $M = \mathsf{TH}(pw, r) \cdot T$ hence we can remove this component of the table TRIC_{pw}, i.e., we now use triples $(r, (s, T))$ and recover M with this equation when needed. As these are just synctactic changes, the games are the same: $P_6 = P_5$.

GAME 7 *(replacing (pw, T) by its H' output)*: Since there are no collisions in the H' table, every (pw, T) pair that appears in a call to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$, or a modification to TRIC_{pw} in $\mathsf{BC.Dec}$, corresponds to a unique k s.t. $k := \mathsf{H}'(pw, T)$[7]. So we can switch the parameters of the table TRIC (and consequently $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$) from (pw, r, T) to (k, r). Besides this, the H queries in $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ can be delayed indefinitely until the adversary actually queries these tables, so we drop these extraneous calls from the definition of BC.

Once again, since we are avoiding our aborts this game change is immaterial and we get $P_7 = P_6$.

GAME 8 *(TRIC is redundant)*: We drop TRIC altogether, since in the previous game it is always copied over to TBC. To be precise, if TRIC_k is not empty - which at this point implies that there exists (a unique) (pw, T) with $\mathsf{TH}'(pw, T) = k$ - then all subsequent (resp. past) accesses and modifications to $\mathsf{TBC}(k, \cdot)$ are (resp. were) done through invoking $\mathcal{F}_{\mathsf{RIC}}$, that is, using TRIC_k. This follows since we are avoiding bckey.abort.

The resulting $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ using TBC directly is presented in Figure 5.9. As these queries are now only made during a $\mathsf{BC.Enc}$ call, we can actually drop AdvEnc usage altogether and expand its definition directly in $\mathsf{BC.Enc}$. The result is that $\mathsf{BC.Enc}$ is simplified into the usual idealized block cipher encryption definition. Likewise, $\mathsf{BC.Dec}$ is also simplified but it

[7]i.e., we could invert such k by $k \mapsto (pw, T)$ where $(pw, T, k) \in \mathsf{TH}'$.

is not yet the idealized block-cipher decryption (see next game). Note that for $\mathsf{BC.Dec}$, the check $r \notin \mathsf{usedR}_{pw}$ implies that there is no $\hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}$. As in the previous games, this is just a syntactic change and $P_8 = P_7$.

```
define F_RIC.AdvEnc(k, r)
if ∄s s.t. (k, r, s) ∈ TBC:
    s ←r {ŝ ∈ {0,1}^n :/
∃r̂ s.t. (k, r̂, ŝ) ∈ TBC}
    add (k, r, s) to TBC
return s
```

Figure 5.9: Replacing usage of TRIC by direct access to TBC

The current full game is given in Figure 5.10.

Game 9 *(real-world)*: In Figure 5.11 we present the real-world game between the environment and our $\mathsf{m2F}$ circuit. This game change consists of dropping the aborts and changing how r is picked in $\mathsf{BC.Dec}$ so as to make TBC consistent with the standard definition of an ideal-cipher. We refer the reader to Figures 5.10 and 5.11.

We start by removing the H' aborts. As before we can bound $|\mathsf{TH}'|$ and $|\mathsf{TBC}|$ by q so that these aborts happen with probability $\leq 2q/2^{\mu}$. H' now matches the real-world definition of Game 9. Then, we modify $\mathsf{BC.Dec}$. We drop the $r \in \mathsf{usedR}_{pw}$ abort in the innermost if and replace the "remove tag $\mathsf{m2F}$..." line with "add r to usedR_{pw}; if it is flagged $\mathsf{m2F}$, remove the flag". We can do so as long as this abort does not happen - i.e. except with probability $\leq |\mathsf{usedR}_{pw}|/2^n \leq q/2^n$. Finally, we compute r as is done in an ideal-cipher even when $k = \mathsf{TH}'(pw, T)$ just as in the real-world. This last change is valid except when our uniform choice of r in Game 8 collides with another $(k, r, \hat{s}) \in \mathsf{TBC}$. This gives us the bound $|\mathsf{TBC}|/2^n \leq q/2^n$. $\mathsf{BC.Dec}$ now matches the real-world in Figure 5.11 except that we have the usedR_{pw} line above.

Now, we are at the real-world game except that we have usedR_{pw} aborts in $\mathsf{m2F.Enc}$, H and

$\mathsf{BC.Enc}$. The probability of the first one is trivially bound by another $|\mathsf{usedR}_{pw}|/2^n \leq q/2^n$ factor. The last two, much like in our argument for the change from Game 0 to Game 1, do not happen except when the adversary is lucky enough to completely guess r since r tagged $\mathsf{m2F}$ is never leaked to the environment. This gives us the following overall bound, which completes the proof:

$$|P_9 - P_8| \leq q^2 \left\{ \frac{2}{2^\mu} + \frac{4}{2^n} \right\} \tag{5.6}$$

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on query $\mathsf{m2F.Enc}(pw, M)$:

$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort
$T \leftarrow M/\mathsf{H}(pw, r)$
$k \leftarrow \mathsf{H}'(pw, T)$
$s \leftarrow \mathsf{BC.Enc}(k, r)$
assign tag $\mathsf{m2F}$ to r in the set usedR_{pw}
return (s, T)

on query $\mathsf{m2F.Dec}(pw, (s, T))$:

$k \leftarrow \mathsf{H}'(pw, T)$
$r \leftarrow \mathsf{BC.Dec}(k, s)$
$M \leftarrow \mathsf{H}(pw, r) \cdot T$
if $\mathsf{m2F.Dec}$ query was fresh, add tag $\mathsf{m2F}$ to $r \in \mathsf{usedR}_{pw}$
return M

on query $\mathsf{H}(pw, r)$

if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$, abort, else add r to usedR_{pw}
if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$:
  $h \xleftarrow{r} \mathbb{G}$
  add (pw, r, h) to TH
return h

on query $\mathsf{H}'(pw, T)$

if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
  $k \xleftarrow{r} \{0,1\}^\mu$
  if $\exists\ (\hat{pw}, \hat{T})$ s.t. $(\hat{pw}, \hat{T}, k) \in \mathsf{TH}'$ then abort (col.abort)
  if $\exists\ (\hat{r}, \hat{s})$ s.t. $(k, \hat{r}, \hat{s}) \in \mathsf{TBC}$ then abort (bckey.abort)
  add (pw, T, k) to TH'
return k

on query $\mathsf{BC.Enc}(k, r)$

if $k = \mathsf{TH}'(pw, T)$:
  if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$, abort, else add $r \in \mathsf{usedR}_{pw}$
if $\nexists\ s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k, \hat{r}, s) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return s

on query $\mathsf{BC.Dec}(k, s)$

if $\nexists\ r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $\exists\ (pw, T)$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
    $r \xleftarrow{r} \{0,1\}^n$
    if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw, T)$
return r

Figure 5.10: Full description of Game 8: one step away from the real-world

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on query $\mathsf{m2F.Enc}(pw, M)$:	on query $\mathsf{m2F.Dec}(pw, (s, T))$:
$r \xleftarrow{r} \{0,1\}^n$ $T \leftarrow M/\mathsf{H}(pw, r)$ $k \leftarrow \mathsf{H}'(pw, T)$ $s \leftarrow \mathsf{BC.Enc}(k, r)$ return (s, T)	$k \leftarrow \mathsf{H}'(pw, T)$ $r \leftarrow \mathsf{BC.Dec}(k, s)$ $M \leftarrow \mathsf{H}(pw, r) \cdot T$ return M
on query $\mathsf{H}(pw, r)$	on query $\mathsf{H}'(pw, T)$
if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$: $h \xleftarrow{r} \mathbb{G}$ add (pw, r, h) to TH return h	if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$: $k \xleftarrow{r} \{0,1\}^\mu$ add (pw, T, k) to TH' return k
on query $\mathsf{BC.Enc}(k, r)$	on query $\mathsf{BC.Dec}(k, s)$
if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$: $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k, \hat{r}, \hat{s}) \in \mathsf{TBC}\}$ add (k, r, s) to TBC return s	if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$: $r \xleftarrow{r} \{\hat{r} \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k, \hat{r}, \hat{s}) \in \mathsf{TBC}\}$ add (k, r, s) to TBC return r

Figure 5.11: Game 9: the real-world interaction between $\mathcal{Z}$ and $\mathsf{m2F}$

By the arguments for indistinguishability of successive games shown above, the total distinguishing advantage of environment $\mathcal{Z}$ between the real-world and the ideal-world interaction is upper-bounded by the following expression, which sums up the bounds given by equations (5.2) to (5.6):

$$|P_0 0 - P_9 9| \leq q^2 \left(\frac{10}{2^n} + \frac{4}{2^n \cdot |\mathbb{G}|} + \frac{6}{2^\mu} \right) \leq q^2 \left(\frac{14}{2^n} + \frac{6}{2^\mu} \right)$$

Since this quantity is negligible, this implies Theorem 5.1 □

Notes on Exact Security. By the above equation, the distinguishability advantage implied by our proof can be upper-bounded as $O(q^2/2^n) + O(q^2/2^\mu)$. Both of these factors are unavoidable in the $\mathsf{m2F}$ construction, and they correspond to collisions in resp. r and k values, either of which allows an attacker to distinguish $\mathsf{m2F}$ from an ideal HIC functionality $\mathcal{F}_{\mathsf{RIC}}$.

Collisions in r values can happen on BC decryption queries, and if the adversary encounters such collision then $\mathsf{m2F}$ fails to act like HIC in the decryption direction. Consider an adversary which computes $k_i = \mathsf{H}'(pw, T_i)$ and $r_i = \mathsf{BC.Dec}(k_i, s_i)$ for a fixed pw and q ciphertexts $(s_1, T_1), ..., (s_q, T_q)$. If $\mathsf{BC.Dec}$ encounters a collision, i.e. $r_i = r_j$ for some i, j, then $\mathsf{m2F}$ decrypts ciphertexts (s_i, T_i) and (s_j, T_j) under the same key pw to plaintexts resp. (r_i, M_i) and (r_i, M_j) s.t. $r_i = r_j$ and $M_i/M_j = T_i/T_j$. HIC decryption can output plaintexts with the same correlations but $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(pw, \cdot)$ on (s_i, T_i) and (s_j, T_j) outputs plaintexts (r, M) s.t. r and M are uncorrelated, while $\mathsf{m2F}^{-1}(pw, \cdot)$ has the property that if $r_i = r_j$ then $M_i/M_j = T_i/T_j$. The adversary can observe these correlations if $\mathsf{BC.Dec}$ collision occurs, which implies distinguishing $\mathsf{m2F}$ from HIC with probability $\Theta(q^2/2^n)$.

Encountering a collision in H' outputs also leads to distinguishing $\mathsf{m2F}$ from HIC. If the adversary finds (pw_1, T_1) and (pw_2, T_2) s.t. $\mathsf{H}'(pw_1, T_1) = \mathsf{H}'(pw_2, T_2)$, then the $s \leftrightarrow r$ maps corresponding to these two pw, T pairs will be the same. For example, for any s, cipher $\mathsf{m2F}$ decrypts ciphertext (s, T_1) under key pw_1 and ciphertext (s, T_2) under key pw_2 to resp. plaintexts (r_1, M_1) and (r_2, M_2) s.t. $r_1 = r_2$. Since HIC decryption outputs independent plaintexts on all decryption queries, this leads to distinguishing $\mathsf{m2F}$ from HIC with probability $\Theta(q^2/2^\mu)$.

Notice that these two terms dominate the advantage of the environment in distinguishing $\mathsf{m2F}$ from the ideal HIC functionality $\mathcal{F}_{\mathsf{RIC}}$. In particular, these terms are independent of group $\mathbb{G}$ and involve only the size of the randomness space $\mathcal{R}$ and the size of the key space of the ideal cipher BC. Note also that the two distinguishing attacks above correspond to abort conditions marked resp. *advdec.abort* and *kcol.abort* in the simulator algorithm SIM in Figure 5.3. The code of simulator SIM has three further abort conditions, and encountering each of them creates inconsistency in the simulation, and can probably be translated into another strategy for distinguishing $\mathsf{m2F}$ and HIC. However, they occur with probabilities upper-bounded by the same bounds $O(q^2/2^n)$ (*rcol.abort* and *advenc.abort*) and $O(q^2/2^\mu)$

($bckey.abort$).

5.4 Encrypted Key Exchange with Randomized Ideal Cipher

We show that the Encrypted Key Exchange (EKE) protocol of Bellovin and Meritt [28] is a universally composable PAKE if the password encryption is implemented with a Randomized (Half-)Ideal Cipher on the domain of messages output by the key exchange scheme, provided that the key exchange scheme has the random-message property (see Section 2). As discussed in the introduction, the same statement was argued by Rosulek et al. [109] with regards to password-encryption implemented using a Programmable Once Public Function (POPF) notion defined therein, which can also be thought of as a weak form of ideal cipher. However, since as we explain in the introduction, the POPF notion is unlikely to suffice in an EKE application, so we need to verify that the notion of UC Randomized (Half-)Ideal Cipher *does* suffice in such application.

In Figure 5.12 we show the Encrypted Key Exchange protocol EKE, specialized to use a Randomized Ideal Cipher for the password-encryption of the message flows of the underlying Key Agreement scheme KA. In Figure 5.12 we assume that KA is a *single-round* scheme. In Section 5.4.1 we extend this to the case of two-flow KA, i.e. to EKE protocol instantiated with a KEM scheme. We note that these two treatments are incomparable because in the case of single-flow KA we start from a more restricted KA scheme and we argue security of a single-flow version of EKE, whereas in the case of two-flow KA, i.e. if $\mathsf{KA} = \mathsf{KEM}$, we start from a more general KA scheme but we argue security of a two-flow version of EKE.

The EKE instantiation shown in Figure 5.12 assumes that the Randomized Ideal Cipher RIC works on domain $\mathcal{D} = \mathcal{R} \times \mathcal{M}$ where $\mathcal{M}$ is the message domain of the scheme KA. The

"randomness" set $\mathcal{R}$ is arbitrary, but its size influences the security bound we show for such EKE instantiations. In particular we require that $\log(|\mathcal{R}|) \geq 2\kappa$. If RIC is instantiated with the modified 2-Feistel construction m2F of Section 5.3, one can set $\mathcal{R} = \{0,1\}^{2\kappa}$, and this instantiation of EKE will send messages whose sizes match those of the underlying KA scheme extended by 2κ bits of randomness due to the Randomized Ideal Cipher encryption.

In Figure 5.12 for presentation clarity we assume that party identifiers $\mathsf{P}_0, \mathsf{P}_1$ are lexicographically ordered. The full protocol will use two helper functions order and bit, defined as $\mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}) = (\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ and $\mathsf{bit}(\mathsf{P}, \mathsf{CP}) = 0$ if $\mathsf{P} <_{lex} \mathsf{CP}$, and $\mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}) = (\mathsf{sid}, \mathsf{CP}, \mathsf{P})$ and $\mathsf{bit}(\mathsf{P}, \mathsf{CP}) = 1$ if $\mathsf{CP} <_{lex} \mathsf{P}$ [8]. Party P on input $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ will then set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ and $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$ and it will use RIC.Enc on key $\hat{pw}_b = (\mathsf{fullsid}, b, pw)$ to encrypt its outgoing message, and it will use RIC.Dec on key $\hat{pw}_{\neg b} = (\mathsf{fullsid}, \neg b, pw)$ to decrypt its incoming message.

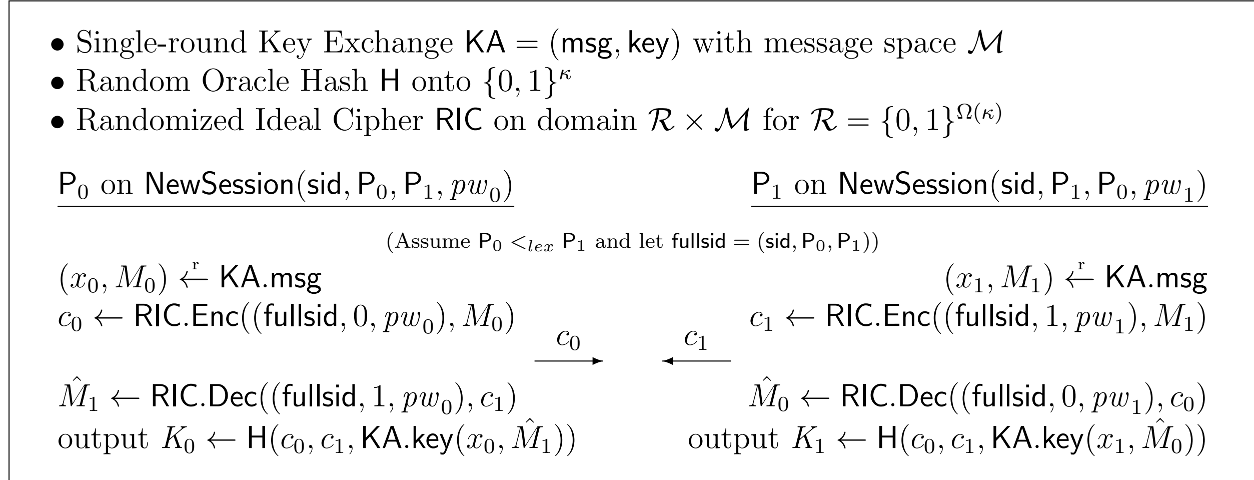


Figure 5.12: EKE: Encrypted Key Exchange with Randomized Ideal Cipher

In Theorem 5.2 below we show that protocol EKE realizes the (multi-session version of) the PAKE functionality of Canetti et al. [48], denoted $\mathcal{F}_{\mathsf{pwKE}}$ (included in Figure 2.1 in Section 2.3.3)(e.g., see [67]). The reason we target the multi-session version of PAKE functionality directly, rather than targeting its single-session version and then resorting to Canetti's composition theorem [47] to imply the security of an arbitrary (and concurrent) number of

[8] We assume that no honest P ever executes $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot)$ for $\mathsf{CP} = \mathsf{P}$.

EKE instances, is that for the latter to work we would need the underlying UC HIC to be instantiated separately for each EKE session identifier sid. Our UC HIC notion of Section 5.2 is a "global" functionality, i.e. it does not natively support separate instances indexed by session identifiers. The modified 2-Feistel construction *could* support such independent instances of HIC by prepending sid to the inputs of all its building block functions $\mathsf{H}, \mathsf{H}', \mathsf{BC}$, where in the last case value sid would have to be prepended to the key of the (ideal) blockcipher BC. However, this implies longer inputs for each of these blocks, which is especially problematic in case of the block cipher, so it is preferable not to rely on it and show security for a protocol variant where each EKE instance accesses a single HIC functionality, and hence can be implemented with the same instantiation of the modified 2-Feistel HIC construction.

Theorem 5.2. *If* KA *is a one-time active secure key-exchange scheme with and the random-message property on domain* $\mathcal{M}$ *and* RIC *is a UC Randomized Ideal Cipher over domain* $\mathcal{R} \times \mathcal{M}$*, then protocol* EKE*, Figure 5.12, realizes the UC PAKE functionality* $\mathcal{F}_{\mathsf{pwKE}}$*.*

Proof. Let $\mathcal{Z}$ be an arbitrary efficient environment. In the rest of the proof we will assume that the real-world adversary $\mathcal{A}$ is an interface of $\mathcal{Z}$. In Figure 5.13 we show the construction of a simulator algorithm SIM, which together with functionality $\mathcal{F}_{\mathsf{pwKE}}$ defines the ideal-world view of $\mathcal{Z}$. As is standard, the role of SIM is to emulate actions of honest parties executing protocol EKE given the information revealed by functionality $\mathcal{F}_{\mathsf{pwKE}}$, and to convert the actions of the real-world adversary into queries to $\mathcal{F}_{\mathsf{pwKE}}$. (In Figure 5.13 we use $\mathsf{P}^{\mathsf{sid}}$ to denote P's session indexed by sid which is emulated by SIM.) The proof then consists of a sequence of games, shown in Figure 5.14, starting from the real-world game, Game 0, where $\mathcal{Z}$ interacts with the honest parties running protocol EKE, and ending with the ideal-world game, Game 7, where $\mathcal{Z}$ interacts via dummy honest parties with functionality $\mathcal{F}_{\mathsf{pwKE}}$ which in turn interacts with simulator SIM. (This last game is not shown in Figure 5.14 because its code can be derived from the code of simulator SIM, Figure 5.13, and functionality $\mathcal{F}_{\mathsf{pwKE}}$, see Figure 2.1.) We note that in each game in Figure 5.14 we write output [...] for output

SIM interacts with environment $\mathcal{Z}$'s interface $\mathcal{A}$ and with functionality $\mathcal{F}_{\mathsf{pwKE}}$. W.l.o.g. we assume that $\mathcal{A}$ uses AdvDec to implement Dec queries to $\mathcal{F}_{\mathsf{RIC}}$.

Initialization: Set $\mathsf{Cset} = \{\}$, set $\mathsf{TRIC}_{\hat{pw}}$ as an empty table and $\mathsf{c2pw}[c] := \perp$ for all values $\hat{pw}$ and c.

Notation (used in all security games in Figure 5.14)

Let $\mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$ be a shortcut for set $\{s \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (s, T)) \in \mathsf{TRIC}_{\hat{pw}}\}$.
Let $\mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$ be a shortcut for set $\{c \in \mathcal{D} : \nexists \hat{m} \text{ s.t. } (\hat{m}, c) \in \mathsf{TRIC}_{\hat{pw}}\}$.
Let $\mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$ be a shortcut for set $\{m \in \mathcal{D} : \nexists \hat{c} \text{ s.t. } (m, \hat{c}) \in \mathsf{TRIC}_{\hat{pw}}\}$.

On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}_{\mathsf{pwKE}}$:

Set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $c \xleftarrow{r} \mathcal{D}$ (*abort* if $c \in \mathsf{Cset}$), add c to Cset, record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, c)$, return c.

Emulating functionality $\mathcal{F}_{\mathsf{RIC}}$:

- On $\mathcal{A}$'s query $(\mathsf{Enc}, \hat{pw}, M)$ to $\mathcal{F}_{\mathsf{RIC}}$: Set $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$. If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return c; Else pick $c \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$ (*abort* if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, return c.

- On $\mathcal{A}$'s query $(\mathsf{AdvEnc}, \hat{pw}, m, T)$ to $\mathcal{F}_{\mathsf{RIC}}$: If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return c; Else pick $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$ (*abort* if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, return c.

- On $\mathcal{A}$'s query $(\mathsf{AdvDec}, \hat{pw}, c)$ to $\mathcal{F}_{\mathsf{RIC}}$: If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return m; Else pick $r \xleftarrow{r} \mathcal{R}$ and $(x, M) \xleftarrow{r} \mathsf{KA.msg}$, set $m \leftarrow (r, M)$, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$ (*abort* if $\exists\, \hat{c} \neq c$ s.t. $(m, \hat{c}) \in \mathsf{TRIC}_{\hat{pw}}$), save $(\mathsf{backdoor}, c, \hat{pw}, x)$, return m.

On $\mathcal{A}$'s message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only the first such message)

Retrieve record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, c)$ and do:

1. If there is record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, \hat{c})$: send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
2. Otherwise set $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and do the following:
   (a) If $\hat{pw} = \perp$ or $\hat{pw} = (\widehat{\mathsf{fullsid}}, \hat{b}, \cdot)$ for $(\widehat{\mathsf{fullsid}}, \hat{b}) \neq (\mathsf{fullsid}, \neg b)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
   (b) If $\hat{pw} = (\mathsf{fullsid}, \neg b, pw^*)$ retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$ and:
      i. service $\mathcal{F}_{\mathsf{RIC}}$'s query $(\mathsf{AdvDec}, (\mathsf{fullsid}, b, pw^*), c)$, retrieve $(\mathsf{backdoor}, c, (\mathsf{fullsid}, b, pw^*), x)$;
      ii. set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$.

Figure 5.13: Simulator SIM for the proof of Theorem 5.2

of queries that service $\mathcal{Z}$'s interaction with EKEinstances, and we write "return [...]" for output of queries that service $\mathcal{Z}$'s interaction with $\mathcal{F}_{\mathsf{RIC}}$.

At each step we prove that the two consecutive games are indistinguishable, which implies the claim by transitivity of computational indistinguishability. Note that we argue security of EKE in the $\mathcal{F}_{\mathsf{RIC}}$-hybrid model. Specifically, algorithm SIM emulates a "global" $\mathcal{F}_{\mathsf{RIC}}$ functionality which services any number of EKE protocol instances. Note that $\mathcal{Z}$ or $\mathcal{A}$ can call $\mathcal{F}_{\mathsf{RIC}}$ on keys which correspond to all strings $\hat{pw} = (\mathsf{fullsid}, b, pw)$ including for fullsid corresponding to sessions which were not (yet) started by $\mathcal{Z}$. Indeed, algorithm SIM treats queries pertaining to any key $\hat{pw}$ equally, and embeds random ciphertext c in response to Enc queries, random partial ciphertext s in response to AdvEnc queries, and random KA message M in response to AdvDec and Dec queries, saving the corresponding KA local state in $(\mathsf{backdoor}, \ldots)$ records. Since Dec is a wrapper over AdvDec we assume that the adversary uses only interface AdvDec, and we implement the EKE code of $\mathsf{P}^{\mathsf{sid}}$ using AdvDec as well.

The intuition for the simulation is that it sends an outgoing EKE message on behalf of $\mathsf{P}^{\mathsf{sid}}$ at random, since this is how RIC encryptions are formed. SIM services RIC encryption queries as $\mathcal{F}_{\mathsf{RIC}}$ does except that it collects the ciphertexts created by any encryption query and the ciphertexts chosen for every honest session in set Cset, and aborts if either process regenerates a ciphertext in Cset. Here we use the fact that even though an adversary can set the T part of the ciphertext $c = (s, T)$ resulting from an adversarial encryption query AdvEnc, the s part of c is chosen at random, and this prevents ciphertext collisions (except with negligible probability) if $|\mathcal{R}| \geq 2^{2\kappa}$. Hence, assuming that $\mathcal{R}$ is big enough, we have that (1) each adversarial ciphertext can be matched to (at most) one password on which it decrypts to a non-random value in space $\mathcal{M}$, and (2) the simulator can extract this unique password and retrieve the corresponding plaintext (SIM stores the key $\hat{pw}$ which was used to create ciphertext c in the c2pw table by setting $\mathsf{c2pw}[c] \leftarrow \hat{pw}$). Moreover, since by the same collision-resistant property of $\mathcal{F}_{\mathsf{RIC}}$ ciphertexts the adversary cannot "hit" any honest

session $\mathsf{P}^{\mathsf{sid}}$'s ciphertext c via an encryption query, the decryption of $\mathsf{P}^{\mathsf{sid}}$'s ciphertext on each password is also a random value in $\mathcal{M}$. By the message-randomness property of KA, simulator SIM can embed messages of fresh KA instances into each decryption query, and combining this with fact (1) above allows for a reduction of EKE instances corresponding to "wrong" password guesses to the KA's security.

Let q_{IC} be the bound on the number of queries $\mathcal{Z}$ makes to the interfaces of the (randomized) ideal cipher $\mathcal{F}_{\mathsf{RIC}}$, and let q_P be the upper-bound on the number of honest EKE sessions $\mathsf{P}^{\mathsf{sid}}$ which $\mathcal{Z}$ invokes for any identifiers P, sid. [9] Let $\varepsilon_{\mathsf{KA.sec}}$ and $\varepsilon_{\mathsf{KA.rand}}$ be the upper-bounds on the distinguishing advantage against, respectively, the security and the random-message properties of the key exchange scheme KA (see Section 2) of an adversary whose computational resources are roughly those of an environment $\mathcal{Z}$ extended by execution of $q_{IC} + q_P$ instances of the key exchange scheme KA.[10]

For a glance of our proof strategy we show the code of all successive games in Figure 5.14. Below we give the full proof for Theorem 5.2.

GAME 0 (real-world game): This is the real world where parties follow the protocol. Technically, it is an *hybrid* world where the randomized ideal cipheris replaced by functionality $\mathcal{F}_{\mathsf{RIC}}$, and the adversary can query $\mathcal{F}_{\mathsf{RIC}}$ through interfaces Enc, AdvEnc, AdvDec.

GAME 1 (randomizing protocol communication): We change the game so ciphertext c sent by $\mathsf{P}^{\mathsf{sid}}$ is purely random, except the game aborts if plaintext (r, M) occurred in table $\mathsf{TRIC}_{\hat{pw}}$ or ciphertext c was output by any encryption. An upper bound on the probability of these aborts is given by

$$|P_0 - P_1| \leq q_P(q_{IC} + q_P)\left(\frac{1}{|\mathcal{R}|} + \frac{1}{|\mathcal{R}| \cdot |\mathcal{M}|}\right) \approx \frac{q_P(q_{IC} + q_P)}{|\mathcal{R}|} \tag{5.7}$$

[9] We assume that $\mathcal{Z}$ invokes at most two sessions for any fixed identifier sid.

[10] This bound involves $q_{IC}+q_P$ instead of q_P key exchange instances because our reductions to KA security run KA.msg for each adversarial AdvDec query to $\mathcal{F}_{\mathsf{RIC}}$.

Game 0: real-world interaction

initialization

Initialize $\mathsf{Cset} = \{\}$ and $\forall\, \hat{pw}$ empty $\mathsf{TRIC}_{\hat{pw}}$

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, b, pw)$
$(x, M) \xleftarrow{r} \mathsf{KA.msg}$
$c \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, M)$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, output c

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one):
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, \cdot, \perp)$:
  $(\hat{r}, \hat{M}) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$
  $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ and output $(\mathsf{sid}, \mathsf{P}, K)$

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, M)$:
$r \xleftarrow{r} \mathcal{R}$, set $m \leftarrow (r, M)$
If $\exists\, c$ s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$:
  return c
else:
  pick $c \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$,
  add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$
  return c

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:
if $\exists\, c$ s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$:
  return c
else:
  $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$,
  add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$
  return c

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:
if $\exists\, m$ s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$:
  return m
else:
  $m \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$
return m

Game 1: randomizing protocol communication

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
set $(\mathsf{fullsid}, b, \hat{pw})$ as in Game 00
$(x, M) \xleftarrow{r} \mathsf{KA.msg}$, $r \xleftarrow{r} \mathcal{R}$, $c \xleftarrow{r} \mathcal{D}$
abort if $((r, M), *) \in \mathsf{TRIC}_{\hat{pw}}$ or $c \in \mathsf{Cset}$
add $((r, M), c)$ to $\mathsf{TRIC}_{\hat{pw}}$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, output c

Game 2: binding adversarial ciphertexts to passwords

on $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, M)$ or $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:
Before adding c to Cset, do the following:
  abort if $c \in \mathsf{Cset}$
  set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$

Game 3: adding trapdoors to decryption

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:

if $\exists m$ s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return m, otherwise:
  $(x, M) \xleftarrow{r} \mathsf{KA.msg}(1^\kappa)$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$
  abort if $(m, *) \in \mathsf{TRIC}_{\hat{pw}}$
  add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$
  save $(\mathsf{backdoor}, c, \hat{pw}, x)$, return m

Game 4: KA messages via AdvDec

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
set $(\mathsf{fullsid}, b, \hat{pw})$ as in Game 00
$c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$
retrieve $(\mathsf{backdoor}, c, \hat{pw}, x)$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, output c

Game 5: extracting passwords

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $\mathsf{rec} = (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, pw, \cdot, \hat{c}, \hat{K})$
  s.t. $\mathcal{Z}$ sent c to $\mathsf{CP}^{\mathsf{sid}}$:
    $K \leftarrow \hat{K}$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$:
      retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$,
      set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$
    else:
      $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, K)$
  output $(\mathsf{sid}, \mathsf{P}, K)$

Game 6: delaying password usage

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$
$c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \perp, c, \perp)$, output c

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \perp, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, pw, \perp, \hat{c}, \hat{K})$:
    $K \leftarrow \hat{K}$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$:
      query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, b, pw), c)$,
      retrieve $(\mathsf{backdoor}, c, \cdot, x)$
      retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$,
      set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$
    else:
      $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, K)$
  output $(\mathsf{sid}, \mathsf{P}, K)$

Figure 5.14: Game changes for the proof of Theorem 5.2 (compare Fig. 5.13 for notation)

GAME 2 (binding adversarial ciphertexts to passwords): We add two changes in the processing of both Enc and AdvEnc queries for any key $\hat{pw}$: If the game responds to either query by picking a new ciphertext c, it (1) aborts if this ciphertext is already in set Cset, (2) otherwise it proceeds but also sets $\mathsf{c2pw}[c] \leftarrow \hat{pw}$. (Initially $\mathsf{c2pw}[c] = \perp$ for all inputs.) The second change is purely syntactic, but the first one introduces a difference upper-bounded by the probability of encountering such collisions. Since Enc picks ciphertext c at random in the space of c's not used for a given key $\hat{pw}$, while AdvEnc picks only the s part of the ciphertext c at random from the space of unused s for a given T and key $\hat{pw}$, the upper-bound on these collisions comes from AdvEnc queries which implies

$$|P_1 - P_2| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}|} \tag{5.8}$$

GAME 3 (embedding KA messages in decryption queries): In this game we embed KA messages into every (fresh) adversarial decryption query $\mathsf{AdvDec}(\hat{pw}, c)$ to $\mathcal{F}_{\mathsf{RIC}}$, and we save the local state generated with this KA message associated with $(c, \hat{pw})$. This change can be thought of as done in two sub-steps: First we change the decryption so it picks (r, M) at random in $\mathcal{R} \times \mathcal{G}$ and aborts if $((r, M), *)$ is in table TRIC. (Note that before a decryption query picks (r, M) according to $\mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$, i.e. among pairs which are not yet in the table.) The difference this introduces is the probability of encountering this abort, which can be upper-bounded as $(q_{IC} + q_P)^2/(|\mathcal{R}| \cdot |\mathcal{M}|)$. The second sub-step is that we pick M according to KA message generation algorithm KA.msg, and we save local state x generated together with M in record $(\mathsf{backdoor}, c, \hat{pw}, x)$. This second change can be reduced to an attack on the random-message property of scheme KA. The argument hybridizes over all decryption queries, where each consecutive hybrid differs by one more decryption query on which M is generated via KA.msg instead of uniform in $\mathcal{G}$. By a reduction to the random-message property of our scheme KA the total difference this change introduces can be upper-bound

as $(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KA.rand}}$. We conclude that:

$$|P_2 - P_3| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}| \cdot |\mathcal{M}|} + (q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KA.rand}} \tag{5.9}$$

GAME 4 (delegating KA message generation to AdvDec): We make a syntactic change in processing $\mathsf{NewSession}$: Rather than picking a random KA message M, random r, a random ciphertext c, and defining $((r, M), c)$ as an IC pair for key $\hat{pw}$, we pick only random c and define M via a decryption query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$. Since in Game 3 a decryption query sets (r, M) in the same way this is only a syntactic change, hence $P_3 = P_4$.

GAME 5 (extracting passwords and randomizing session keys on "wrong" passwords): We change how $\mathsf{P}^{\mathsf{sid}}$ reacts to a received ciphertext $\hat{c}$. First of all we introduce a special processing in case $\hat{c}$ is sent by a matching session $\mathsf{CP}^{\mathsf{sid}}$ (i.e. a session that uses the same sid, same pw, and matching P, CP values) that received the honest message c sent out by $\mathsf{P}^{\mathsf{sid}}$. In this case we short-cut all processing and simply set the session key output by $\mathsf{P}^{\mathsf{sid}}$ to the one which was output by $\mathsf{CP}^{\mathsf{sid}}$. Note this corresponds to the case where the environment does not interfere in the communication between $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$. This introduces no change because such sessions compute the same session keys in all previous games. Secondly, for all other $\hat{c}$ cases, instead of using $\mathcal{F}_{\mathsf{RIC}}$ to decrypt $\hat{c}$ under stored $\hat{pw}$, and using the decrypted plaintext $\hat{M}$ to compute the session key as $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, we set $\hat{pw} = \mathsf{c2pw}[\hat{c}]$ and consider two cases: If $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$ then Game 5 computes K in the same way as in Game 4, except that we render the decryption query as a retrieval from table $\mathsf{TRIC}_{\hat{pw}}$ instead of as a query to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$, but this is only a notational change; In any other case, Game 5 shortcuts this decryption and key-computation process and outputs a random key $K \xleftarrow{\mathrm{r}} \{0,1\}^{\kappa}$.

The argument that Game 4 is indistinguishable from Game 5 is a hybrid argument which changes the view in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ invoked by $\mathcal{Z}$. Note that the only case where there is a difference between the two games is the last one we de-

scribed, i.e. if $\mathsf{c2pw}[\hat{c}]$ contains an entry $\hat{pw} \neq (\mathsf{fullsid}, \neg b, pw)$.[11] This corresponds to two sub-cases: (a) $\hat{c}$ was created via an adversarial encryption query on some key $\hat{pw}$ which does not match the decryption key $(\mathsf{fullsid}, \neg b, pw)$ that $\mathsf{P}^{\mathsf{sid}}$ would use in Game 4 to decrypt this ciphertext (note that this $\hat{pw}$ is unique because of an abort in the case two encryption queries ever create the same ciphertext); and (b) $\hat{c}$ was not created in any encryption query. In either of these two sub-cases Game 4 would compute $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ for $\hat{M} \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$, and since in either case $\hat{c}$ was not inserted in table $\mathsf{TRIC}_{(\mathsf{fullsid}, \neg b, pw)}$ via an encryption query, this AdvDec query will embed a random KA message into the decrypted plaintext $\hat{M}$.

We will argue that the existence of an adversary who distinguishes with non-negligible advantage between Games 4 and 5 implies an attack on the security property of the key exchange scheme KA. Since message M created by $\mathsf{P}^{\mathsf{sid}}$ is a random KA message, and we argued above that $\hat{M} = \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$ is a random KA message as well, the session key K which $\mathsf{P}^{\mathsf{sid}}$ outputs in this case in Game 4 is a KA output on an exchange involving two random KA messages, M and $\hat{M}$. It can thus be replaced by a random string by a reduction to KA security done separately for each sid, in two sub-steps corresponding to the (at most) two sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ which run on this particular sid. Consider the argument for a fixed session $\mathsf{P}^{\mathsf{sid}}$ with corresponding counterparty session $\mathsf{CP}^{\mathsf{sid}}$ and $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$: Given the KA security challenge $(M, \hat{M}, K)$, the reduction does the following: First, it uses challenge value M when computing the outgoing message c of $\mathsf{P}^{\mathsf{sid}}$, i.e. it uses M from the challenge when processing query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ in Game 4. Second, it guesses an index $i \xleftarrow{r} [1, ..., q_{IC} + q_P]$ of a query to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ using key $(\mathsf{fullsid}, \neg b, pw)$ and embeds challenge value $\hat{M}$ into the decrypted plaintext. (Note that by Game 4 each $\mathsf{NewSession}$ query also uses AdvDec. Notice also that $\hat{c}$ in this AdvDec query could be equal to ciphertext c' generated by session $\mathsf{CP}^{\mathsf{sid}}$, this corresponds to the adversary passively connecting two sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ which run on matching inputs). Third, if the guess is right and the adversary sends

[11]This includes the case of $\hat{pw} = \perp$.

ciphertext $\hat{c}$ used in this i-th query to $\mathsf{P}^{\mathsf{sid}}$, the reduction embeds the K challenge value into the session key output by $\mathsf{P}^{\mathsf{sid}}$ (if the guess is not right the reduction aborts). If the guess is right and case (b) occurs, the reduction reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 4 if K is the real key corresponding to KA instance $(M, \hat{M})$, and it reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 5 if K is random. Since the right guess occurs with probability $1/(q_{IC}+q_P)$ and the identity of the index i does not affect the view the reduction produces before the abort, and the argument goes by a hybrid over all honest party sessions, we arrive at the following upper-bound:

$$|P_4 - P_5| \leq \frac{(q_{IC}+q_P)^2}{|\mathcal{R}| \cdot |\mathcal{M}|} + q_P(q_{IC}+q_P) \cdot \varepsilon_{\mathsf{KA.sec}} \tag{5.10}$$

GAME 6 (delaying password usage): In this game we delay using the password pw of session $\mathsf{P}^{\mathsf{sid}}$ to decrypt (and consequently embed the backdoor into) its honest outgoing message c to the moment $\mathsf{P}^{\mathsf{sid}}$ receives an incoming message $\hat{c}$. Moreover, we perform this decryption only in the case adversary created $\hat{c}$ via encryption under key $(\mathsf{fullsid}, \neg b, pw)$. Since Game 5 does not use the decrypted value M and the associated trapdoor x until this exact situation occurs, postponing this decryption does not matter as long as item $(*, c)$ is not written into table $\mathsf{TRIC}_{(\mathsf{fullsid},b,pw)}$ via an encryption query. However, the latter cannot happen in Game 5 because each $\mathsf{NewSession}$ and each encryption queries generate disjoint ciphertexts (a collision in the ciphertexts created by any of these queries leads to an abort), which implies that $P_5 = P_6$.

GAME 7 (ideal-world game implied by $\mathcal{F}_{\mathsf{pwKE}}$ and SIM): This is the ideal-world game induced by functionality $\mathcal{F}_{\mathsf{pwKE}}$ interacting with simulator SIM of Figure 5.13. In that interaction $\mathcal{F}_{\mathsf{pwKE}}$ creates a session record with password pw in it, but $\mathcal{F}_{\mathsf{pwKE}}$ does not pass pw to SIM. However, SIM picks $\mathsf{P}^{\mathsf{sid}}$'s c at random and aborts if $c \in \mathsf{Cset}$, which is how $\mathsf{NewSession}$ processing is done in Game 6. Note also that SIM replies to Enc or AdvEnc queries in a

way which matches processing of these queries starting from Game 2, and that it replies to AdvDec queries in a way which matches processing of these queries starting from Game 3. Finally, when the environment sends $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$, we have the following cases:

1. Message $\hat{c}$ was sent by counterparty session $\mathsf{CP}^{\mathsf{sid}}$ which matches session $\mathsf{P}^{\mathsf{sid}}$ in session identifier sid and party identifiers $(\mathsf{P}, \mathsf{CP})$. This case is detected by the simulator SIM who can check if identifiers $(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ of the two sessions match, and it corresponds to step 1 in SIM's processing of $\hat{c}$. In this case SIM sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \bot)$ to $\mathcal{F}_{\mathsf{pwKE}}$ in which case $\mathcal{F}_{\mathsf{pwKE}}$, since this NewKey was *not* proceeded by a $\mathsf{TestPwd}$ so session $\mathsf{P}^{\mathsf{sid}}$ is marked fresh, does either of the following two things: (case 1) if the two sessions run on the same password and $\mathsf{CP}^{\mathsf{sid}}$ completed while marked fresh, which happens only if the adversary sent (to $\mathsf{CP}^{\mathsf{sid}}$) the unmodified ciphertext c output by $\mathsf{P}^{\mathsf{sid}}$, then $\mathcal{F}_{\mathsf{pwKE}}$ makes key K output by $\mathsf{P}^{\mathsf{sid}}$ equal to key $\hat{K}$ output by $\mathsf{CP}^{\mathsf{sid}}$; and (case 2) in any other case $\mathcal{F}_{\mathsf{pwKE}}$ picks key K output by $\mathsf{P}^{\mathsf{sid}}$ at random.

   Note that this is exactly how Game 6 processes delivery of $\hat{c}$ output by $\mathsf{CP}^{\mathsf{sid}}$ as well. Case 1 corresponds to the first check performed by $\hat{c}$-delivery processing code of Game 6 which assigns $K \leftarrow \hat{K}$ if all inputs of $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ match and the adversary delivered the ciphertext output by $\mathsf{P}^{\mathsf{sid}}$ to $\mathsf{CP}^{\mathsf{sid}}$. Case 2 means that the $\hat{c}$-delivery processing code of Game 6 will recover $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and check if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$. In case 2, where $\hat{c}$ is output by $\mathsf{CP}^{\mathsf{sid}}$ value $\mathsf{c2pw}[\hat{c}]$ is guaranteed to be $\bot$ because Game 6, just like the ideal-world interaction, does not allow collisions between ciphertexts output by honest sessions and ciphertexts output via Enc or AdvEnc queries. Therefore $\hat{c}$-delivery processing code of Game 6 will jump to the second "else" clause and set $K \xleftarrow{r} \{0,1\}^{\kappa}$, matching the behavior of the ideal-world interaction.

2. If message $\hat{c}$ was *not* sent by counterparty session $\mathsf{CP}^{\mathsf{sid}}$ which matches session $\mathsf{P}^{\mathsf{sid}}$ in its session+party identifiers inputs, i.e. if $\hat{c}$ is a ciphertext created by the adversary (or output by any other session than the intended counterparty of $\mathsf{P}^{\mathsf{sid}}$), this corresponds

to 2. in SIM's definition of its processing of $\hat{c}$, which has two sub-cases based on the value $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$:

(a) In (a) SIM processes the case when $\hat{pw} = \perp$ or $\hat{pw} \neq \perp$ but $\hat{pw}$ does not have the form $(\mathsf{fullsid}, \neg b, pw^*)$ for any password pw^* (which means that $\hat{pw}$ is guaranteed not to match the key $\mathsf{P}^{\mathsf{sid}}$ would use to decrypt $\hat{c}$ regardless of the password $\mathsf{P}^{\mathsf{sid}}$ uses). In that case SIM sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$ before sending $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$, which means that $\mathcal{F}_{\mathsf{pwKE}}$ marks this session as $\mathsf{interrupted}$ and sets its key as $K \xleftarrow{r} \{0,1\}^\kappa$.

Observe that in this case Game 6 will set K in the same way as in the above $\mathsf{SIM}+\mathcal{F}_{\mathsf{pwKE}}$ interaction, because $\hat{pw} \neq (\mathsf{fullsid}, \neg b, pw^*)$ for any pw^* including pw held by $\mathsf{P}^{\mathsf{sid}}$, so the $\hat{c}$-delivery processing code of Game 6 will go to the second "else" clause and set $K \xleftarrow{r} \{0,1\}^\kappa$.

(b) In (b) SIM processes the case when $\hat{pw} = (\mathsf{fullsid}, \neg b, pw^*)$ for some pw^*, which might or might not be equal to the password input pw of $\mathsf{P}^{\mathsf{sid}}$. In this case SIM retrieves $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$, services query $(\mathsf{AdvDec}, (\mathsf{fullsid}, b, pw^*), c)$, retrieves $(\mathsf{backdoor}, c, (\mathsf{fullsid}, b, pw^*), x)$, sets $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, and sends ($\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$. Consider two sub-cases depending on $\mathsf{P}^{\mathsf{sid}}$'s input pw:

i. If $pw^* \neq pw$ then $\mathcal{F}_{\mathsf{pwKE}}$ will mark session $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{interrupted}$ in response to the above $\mathsf{TestPwd}$ query, and consequently $\mathcal{F}_{\mathsf{pwKE}}$ will ignore the value K which SIM sends in the NewKey query, and it will pick the session key output by $\mathsf{P}^{\mathsf{sid}}$ uniformly from $\{0,1\}^\kappa$.

   This is also how Game 6 $\hat{c}$-delivery code will process this case, because it corresponds to the case when $\hat{pw}$ retrieved from $\mathsf{c2pw}[\hat{c}]$ is not equal to $(\mathsf{fullsid}, \neg b, pw)$.

ii. If $pw^* = pw$ then $\mathcal{F}_{\mathsf{pwKE}}$ will mark session $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{compromised}$ in response

to $\mathsf{TestPwd}$ and in response to NewKey it will make $\mathsf{P}^{\mathsf{sid}}$ output the key K computed by SIM.

This is also how Game 6 will behave in this case, because it corresponds to the case $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$, in which case Game 6 retrieves backdoor x as the KA state corresponding to the decryption of c under key $(\mathsf{fullsid}, b, pw)$, and it sets $\mathsf{P}^{\mathsf{sid}}$'s output as $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ for $((\hat{r}, \hat{M}), \hat{c})$ retrieved from $\mathsf{TRIC}_{\hat{pw}}$, exactly like SIM does above.

Since Game 6 matches the ideal-world interaction of Game 7 exactly we conclude that $P_6 = P_7$, which completes the proof of Theorem 5.2.

From the proof the total distinguishing advantage of environment $\mathcal{Z}$ between the real-world and the ideal-world interaction is upper-bounded by the following expression, which sums up the bounds given by equations (5.7), (5.8), (5.9), (5.10):

$$(q_{IC} + q_P) \left[\frac{1}{|\mathcal{R}|} \cdot \left\{ 2q_P + q_{IC} + 2 \cdot \frac{q_{IC} + q_P}{|\mathcal{M}|} \right\} + \varepsilon_{\mathsf{KA.rand}} + q_P \cdot \varepsilon_{\mathsf{KA.sec}} \right] \tag{5.11}$$

Since this quantity is negligible if $\mathcal{R} = \{0,1\}^{\Omega(\kappa)}$, it implies Theorem 5.2. □

Notes on Exact Security.

The dominating factors are $(q_{IC} + q_P)^2/|\mathcal{R}|$ and $(q_{IC} + q_P) \cdot (\varepsilon_{\mathsf{KA.rand}} + q_P \cdot \varepsilon_{\mathsf{KA.sec}})$. The first factor is due to possible collisions in Randomized Ideal Cipher, and it is unavoidable using an arbitrary HIC realization because it is the probability of generating the same ciphertext c as an encryption of two different KA instances under two different passwords, which would also form an explicit attack on the security of EKE (the adversary would effectively make two password guesses in one on-line interaction). However, whereas the bound $(q_{IC})^2/|\mathcal{R}|$ is tight if the encryption is modeled as a Randomized Ideal Cipher, we do not know if it is tight in relation to the specific modified 2-Feistel instantiation of Randomized Ideal Cipher, because

we do not know how to stage an explicit attack on EKE using modified 2-Feistel along these lines. This relates to the fact that whereas the modified 2-Feistel realizes functionality $\mathcal{F}_{\mathsf{RIC}}$, this functionality allows more freedom to the adversary than the modified 2-Feistel construction. Namely, whereas $\mathcal{F}_{\mathsf{RIC}}$ allows the adversary to encrypt any messages M using a ciphertext $c = (s, T)$ where T can be freely set, the same is not true about the modified 2-Feistel construction, where for any fixed M the adversary can choose T from the set of values of the form $T = M/\mathsf{H}(pw, r)$ for some r.

The second factor is due to reductions to KA security properties. Note that some KA schemes, e.g. Diffie-Hellman, have perfect message-randomness, i.e. $\varepsilon_{\mathsf{KA.rand}} = 0$. Further, if the KA scheme is *random self-reducible*, as is Diffie-Hellman, then this factor can be reduced to $\varepsilon_{\mathsf{KA.sec}}$ because a reduction to KA security for the transition between Games 44 and 55, see Section 5.4 the proof in [67], can then be modified so that it deals with all honest sessions at once instead of staging a hybrid argument over all sessions, and it embeds randomized versions of the KA challenge into each decryption query rather than guessing a target query.

5.4.1 EKE with Randomized Ideal Cipher : the KEM version

In Figure 5.15 we show protocol $\mathsf{EKE\text{-}KEM}$, which is a KEM version of the EKE protocol using a Randomized Ideal Cipher. In the 1-flow protocol EKE considered in Figure 5.12, the message flows are generated by a single-round KA scheme, whereas here we consider an EKE variant which is built from any two-flow key exchange, i.e. KEM, see Section 2.2. The drawback is that it is 2-flow instead of 1-flow, but the benefits are that the HIC can be used only for one message, so if KEM is instantiated with Diffie-Hellman and HIC is implemented using $\mathsf{m2F}$, this implies a single RO hash onto a group per party instead of two such hashes. Moreover, this version of EKE can use any CPA-secure KEM as a black box, as long as the KEM satisfies the anonymity and uniform public keys properties, which implies, e.g.,

lattice-based UC PAKE given any lattice-based KEM with these properties.

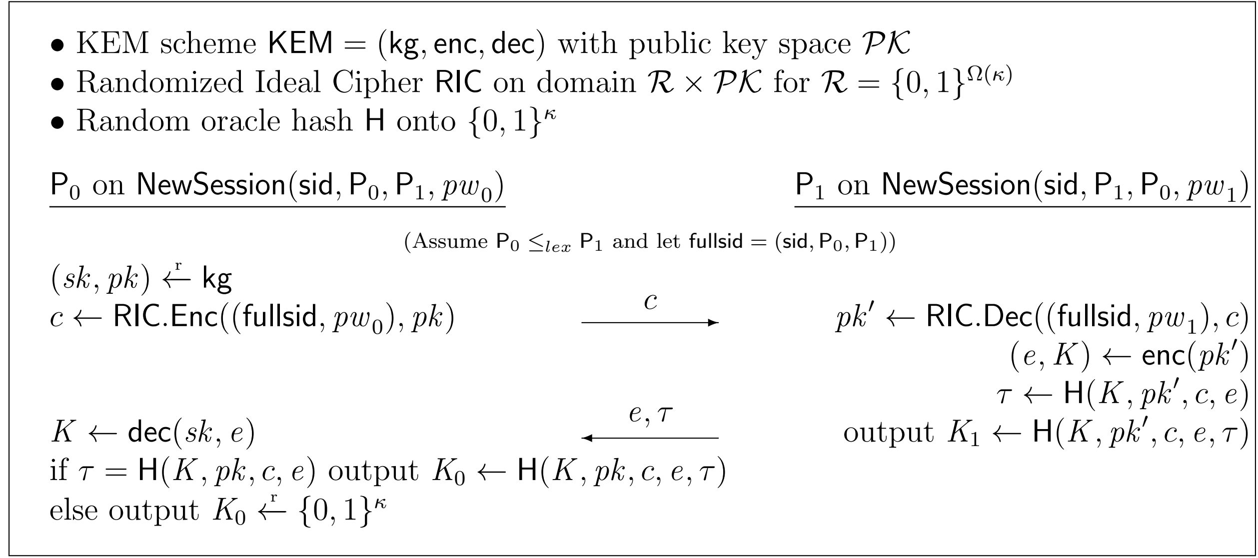


Figure 5.15: $\mathsf{EKE\text{-}KEM}$: Encrypted Key Exchange with Randomized Ideal Cipher (KEM version)

Note that in the protocol of Fig. 5.15 party P_0 outputs a random session key if the key confirmation message τ fails to verify. This is done only so that the protocol conforms to the implicit-authentication functionality $\mathcal{F}_{\mathsf{pwKE}}$. In practice P_0 could output $\perp$ in this case, and this would implement explicit authentication in the P_1-to-P_0 direction.

Theorem 5.3. *If* KEM *is IND secure, anonymous, and has uniform public keys in domain* $\mathcal{PK}$ *(see Section 2.2),* RIC *is a UC Randomized Ideal Cipher in domain* $\mathcal{R} \times \mathcal{PK}$*, and* H *is an RO hash, then protocol* $\mathsf{EKE\text{-}KEM}$ *realizes the UC PAKE functionality* $\mathcal{F}_{\mathsf{pwKE}}$.

The proof of Theorem 5.3 follows the same blueprint as the proof of Theorem 5.2. The most important intuition needed for the adaptation of the proof of Theorem 5.2 to the proof of Theorem 5.3 is why it works for KEMs that satisfy the anonymity property: The key issue is that we need anonymity of the KEM ciphertext e only for honest keys pk and not for adversarial ones, and the reason for this is that the only non-random pk under which an honest party encrypts is the key pk decrypted under a unique password guess pw^* used in the adversarial ciphertext c this party receives. If pw^* equals to P_1's password pw then this session is already successfully attacked, so the non-randomness of P_1's ciphertext is

not an issue. But if $pw^* \neq pw$ then KEM ciphertext e is effectively encrypted under key $pk' = \mathsf{AdvDec}(pw, c)$ which is random, and the key confirmation works as a commitment to the KEM key pk decrypted from HIC ciphertext c, hence also to the password used in that decryption. This commitment is also effectively encrypted under the KEM session key K, hence it can be verified only by a party which created pk and HIC-encrypted it under the right pw. Here we again rely on the property of HIC, which just like IC assures that decryption under any password except for the unique password committed in the ciphertext results in a random plaintext, i.e. a random KEM public key pk, which makes the KEM session key K encrypted under such pk hidden to the adversary by KEM security.

We note that the key confirmation could involve directly pw instead of pk, but pk is a commitment to pw unless the adversary creates a collision in HIC plaintext, and using pk instead of pw lets P_0 erase pw after sending its first message. This way an adaptive compromise on party P_0 during protocol execution allows for offline dictionary attack on the password, but does not leak it straight away. (Note that adaptive party compromise is not part of our security model.) We note also that RO hash H can probably be replaced by a key derivation function which is both a CRH (because it needs to commit to pk) and a PRF (because it must encrypt this commitment under K), but since HIC implies RO hash (and indeed our m2Fuses it) we opt for the simpler option of RO hash to compute the authenticator.

Below we prove Theorem 5.3 from Section 5.4.1, i.e. the EKE-KEM protocol shown in Figure 5.15 is a UC PAKE. Following the blueprint of EKE proof in Section 5.4, we argue security of EKE-KEM in the $\mathcal{F}_{\mathsf{RIC}}$-hybrid model.

Figure 5.16 shows simulator SIM used in this proof. We fix the adversarial environment $\mathcal{Z}$, and let $q_{\mathsf{IC}}, q_{\mathsf{H}}, q_{\mathsf{NS}}$ be the maximum bounds on resp. $\mathcal{Z}$'s queries to the HIC interfaces $\mathcal{F}_{\mathsf{RIC}}$, $\mathcal{Z}$'s queries to the RO hash H, and $\mathcal{Z}$'s calls to NewSession to invoke honest protocol parties. Let $\varepsilon_{\mathsf{KEM.sec}}$, $\varepsilon_{\mathsf{KEM.randpk}}$, and $\varepsilon_{\mathsf{KEM.anonymity}}$ be the upper-bounds on the distinguishing

advantage against respectively the IND security, the uniform public keys, and the anonymity properties of KEM (see Section 2.2), of an attacker whose computational resources are comparable to those of $\mathcal{Z}$, plus the cost of our reductions, which are always close to the cost of simulator SIM running against $\mathcal{Z}$.

Simulator Notation. We write simulator SIM in Figure 5.16 s.t. it interacts with environment $\mathcal{Z}$'s "adversary" interface $\mathcal{A}$, and with PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$. Let $\mathcal{PK}$ be the public key space of KEM, and let $\mathcal{D} = \mathcal{R} \times \mathcal{PK}$ be the domain of Randomized Ideal Cipher RIC. Without loss of generality we assume that $\mathcal{A}$ uses interface AdvDec to implement Dec query to $\mathcal{F}_{\mathsf{RIC}}$. We use $\mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, $\mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$ and $\mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$ as shortcuts for respectively sets $\{s \in \mathcal{R} \text{ s.t. } (\cdot,(s,T)) \notin \mathsf{TRIC}_{\hat{pw}}\}$, $\{m \in \mathcal{D}: \text{ s.t. } (m,\cdot) \notin \mathsf{TRIC}_{\hat{pw}}\}$ and $\{c \in \mathcal{D}: \text{ s.t. } (\cdot,c) \notin \mathsf{TRIC}_{\hat{pw}}\}$.

Proof. Game 0 (real-world game): This is the real world constructed by parties following the protocol, functionality $\mathcal{F}_{\mathsf{RIC}}$, and the adversary who can query $\mathcal{F}_{\mathsf{RIC}}$ through interfaces $\mathsf{Enc}, \mathsf{AdvEnc}, \mathsf{AdvDec}$. We also record all ciphertexts generated by $\mathsf{Enc}, \mathsf{AdvEnc}$ queries into set Cset which is syntactic.

Game 1 (randomizing first message): In this game we change how $\mathsf{P}^{\mathsf{sid}}$ generates the first message c. As in Game 0, $\mathsf{P}^{\mathsf{sid}}$ picks $(sk, pk) \xleftarrow{r} \mathsf{KEM.kg}$, but then instead of querying c via $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}$, Game 1 picks c at random, and pick a random r. And before adding $((r,pk),c)$ to $\mathsf{TRIC}_{\hat{pw}}$ and adding c to Cset, the game aborts (1) if $((r,pk),\cdot)$ is already in table $\mathsf{TRIC}_{\hat{pw}}$, or (2) if ciphertext c randomly picked by P_0 in Game 1 was output by any previous encryption queries, i.e. $c \in \mathsf{Cset}$. The probability of these aborts is upper-bounded by $\frac{q_P(q_{IC}+q_P)}{|\mathcal{R}|}$.

We further bind adversarial ciphertexts to passwords by removing collisions on ciphertexts generated (for any key $\hat{pw}$) by Enc, AdvEnc queries. We add an abort if any new c generated by such queries already exists in Cset. Since by definition Enc query already picks c at

Notation: See the "Simulator Notation" note in Section 5.4.1.
Initialization: Set $\mathsf{Cset} = \{\}$, set $\mathsf{T_H}$ as empty table, set $\mathsf{TRIC}_{\hat{pw}}$ as empty table for all $\hat{pw}$, set $\mathsf{c2pw}[c] := \perp$ for all c.
On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}_{\mathsf{pwKE}}$:
Set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$.

1. If $b = 0$, pick $c \xleftarrow{r} \mathcal{D}$ (*abort* if $c \in \mathsf{Cset}$), add c to Cset, send c to $\mathcal{A}$ as a message from $\mathsf{P}^{\mathsf{sid}}$ and record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 0, \mathsf{fullsid}, c)$.
2. If $b = 1$, record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \perp, \perp, \perp)$.

Emulating $\mathcal{F}_{\mathsf{RIC}}$:

- On $\mathcal{A}$'s query $(\mathsf{Enc}, \hat{pw}, M)$ to $\mathcal{F}_{\mathsf{RIC}}$: Set $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$. If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return c; Else pick $c \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$ (*abort* if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvEnc}, \hat{pw}, m, T)$ to $\mathcal{F}_{\mathsf{RIC}}$: If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return c; Else pick $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$ (*abort* if $c \in \mathsf{Cset}$), $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvDec}, \hat{pw}, c)$ to $\mathcal{F}_{\mathsf{RIC}}$: If $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return m; Else pick $r \xleftarrow{r} \mathcal{R}$ and $(pk, sk) \xleftarrow{r} \mathsf{KEM.kg}$, set $m \leftarrow (r, pk)$ (*abort* if $(m, \cdot) \in \mathsf{TRIC}_{\hat{pw}}$), add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, save $(\mathsf{backdoor}, c, \hat{pw}, sk, pk)$, return m.

On $\mathcal{A}$'s message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only one such message)
Retrieve record $\mathsf{rec} = (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \perp, \perp, \perp)$, pick $\tau \xleftarrow{r} \{0,1\}^\kappa$, $pk^* \xleftarrow{r} \mathcal{PK}$, and $(e, \cdot) \leftarrow \mathsf{KEM.enc}(pk^*)$, and do the following:

- If $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, 0, \mathsf{fullsid}, \hat{c})$ then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$.
- Otherwise, set $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and if $\hat{pw} = \perp$ or $\hat{pw} \neq (\mathsf{fullsid}, \cdot)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
- Otherwise, i.e. if $\hat{pw} = (\mathsf{fullsid}, pw^*)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ to $\mathcal{F}_{\mathsf{pwKE}}$ and:
  1. if answer is "incorrect", send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
  2. if answer is "correct", retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$, reset $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$ and $\tau \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e)$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \mathsf{H}(K^*, \hat{pk}, \hat{c}, e, \tau))$ to $\mathcal{F}_{\mathsf{pwKE}}$.

Update rec to $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \hat{c}, e, \tau)$ and send (e, τ) to $\mathcal{A}$.
On $\mathcal{A}$'s message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only one such message)
Retrieve record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 0, \mathsf{fullsid}, c)$ and:

- If $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, 1, \mathsf{fullsid}, c, \hat{e}, \hat{\tau})$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$.
- Otherwise, if $\exists$ record $(\mathsf{backdoor}, c, (\mathsf{fullsid}, pw^*), sk, pk)$ s.t. $\hat{\tau} = \mathsf{H}(K^*, pk, c, \hat{e})$ for $K^* = \mathsf{KEM.dec}(sk, \hat{e})$ (*abort* if multiple pw^* satisfy above), send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \mathsf{H}(K^*, pk, c, \hat{e}, \hat{\tau}))$ to $\mathcal{F}_{\mathsf{pwKE}}$
- Otherwise send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$

On $\mathcal{A}$'s query x to H:
If $\exists \langle x, y \rangle$ in $\mathsf{T_H}$ output y, else output $y \xleftarrow{r} \{0,1\}^\kappa$ and add $\langle x, y \rangle$ to $\mathsf{T_H}$.

Figure 5.16: Simulator SIM for the proof of Theorem 5.3

random in the space of unused c's for a given $\hat{pw}$, the game aborts only in AdvEnc which picks the s part of c for a given T and $\hat{pw}$. The probability of encountering abort is upper-bounded by $\frac{q_{IC}^2}{|\mathcal{R}|}$. We also add a syntactic change where we record $\mathsf{c2pw}[c] \leftarrow \hat{pw}$ in Enc or AdvEnc queries. As in the case of EKE proof we have:

$$|P_0 - P_1| \leq \frac{q_P(q_{IC} + q_P)}{|\mathcal{R}|} + \frac{q_{IC}^2}{|\mathcal{R}|} \tag{5.12}$$

Game 2 (embedding public key in decryption queries): In this game we embed public key into every fresh adversarial $\mathsf{AdvDec}(\hat{pw}, c)$ query to $\mathcal{F}_{\mathsf{RIC}}$, and we save the corresponding (sk, pk) associated with $(c, \hat{pw})$. This change can be done in two sub-steps: First we change the decryption so it picks (r, pk) randomly in $\mathcal{R} \times \mathcal{PK}$, and aborts if $((r, pk), \cdot)$ is in table $\mathsf{TRIC}_{\hat{pw}}$, whereas before, a decryption query picks (r, pk) according to $\mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$, i.e. among pairs which are not yet in the table. The probability of encountering this abort can be upper-bounded by $(q_{IC} + q_P)^2/(|\mathcal{R}| \cdot |\mathcal{PK}|)$. The second sub-step is that, instead of picking pk at random, we generate key pair (sk, pk) according to KEM key generation algorithm kg, and we save (sk, pk) in record $(\mathsf{backdoor}, c, \hat{pw}, sk, pk)$. This second change can be reduced to an attack on the uniform public keys property 2.5 of KEM. The argument hybridizes over all decryption queries, where each consecutive hybrid differs by one more decryption query on which pk is generated via kg instead of uniform in $\mathcal{PK}$. By a reduction to the uniform public keys property of KEM the total difference this change introduces can be upper-bound as $(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.randpk}}$. We conclude that:

$$|P_1 - P_2| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}| \cdot |\mathcal{PK}|} + (q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.randpk}} \tag{5.13}$$

Game 3 (delegating key generation to AdvDec): We make a syntactic change in processing $\mathsf{NewSession}$ for the party $\mathsf{P}^{\mathsf{sid}}$ who sends out message c: rather than generating random

(sk, pk), r, c, and adding $((r, pk), c)$ as an IC pair for key $\hat{pw}$ in Game 1, Game 3 picks c at random(abort if $c \in \mathsf{Cset}$) as before, but then it retrieves (sk, pk) via a decryption query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$. Since in Game 2 such a decryption query sets (sk, pk) and adds $((r, pk), c)$ to $\mathsf{TRIC}_{\hat{pw}}$ in the same way, this is only a syntactic change, hence $P_2 = P_3$.

Game 4 (abort on H collision):

We add an abort on H collisions. It follows that $|P_4 - P_3| \leq \frac{q_H^2}{2^\kappa}$

Game 5 (randomizing second message and session keys in passive cases):

In this game we change how $\mathsf{P}^{\mathsf{sid}}$ reacts to received message $\hat{c}$ in the passive case, where $\mathsf{P}^{\mathsf{sid}}$ receives the honest message $\hat{c}$ sent by a matching session $\mathsf{CP}^{\mathsf{sid}}$. In this case we shortcut all processing, and let $\mathsf{P}^{\mathsf{sid}}$ generate e via KEM encapsulation on a random public key picked by SIM, and output τ and session key K as random elements in $\{0,1\}^\kappa$. Furthermore, if $\mathsf{CP}^{\mathsf{sid}}$ receives this (e, τ) then we shortcut all processing and simply set the session key of $\mathsf{CP}^{\mathsf{sid}}$ to K output by $\mathsf{P}^{\mathsf{sid}}$. Note this corresponds to the case where the environment does not interfere in the communication between $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$. We abort if τ has been output by H before, thus the change on τ introduces the difference bounded by $\frac{q_H}{2^\kappa}$. The change on e can be reduced to an attack on the anonymity property 2.6 of KEM. The argument is hybrid and changes in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ who received this passive $\hat{c}$. Each consecutive differs by one more e generation, where e is picked via $\mathsf{KEM.enc}(pk^*)$ for a random pk^* picked by SIM, instead of generated via $\mathsf{KEM.enc}(pk)$ for $pk \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$ where $\hat{pw}$ refers to the pw which $\mathsf{P}^{\mathsf{sid}}$ holds. By a reduction to the anonymity property of KEM, the total difference introduced by this change can be upper-bounded as $(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.anonymity}}$. The change on session key generation introduces no difference because such sessions compute same session keys in all previous games. Thus we have $|P_5 - P_3| \leq \frac{q_H}{2^\kappa} + (q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.anonymity}}$

Game 6 (randomizing second message and session keys in other cases):

Now we change how $\mathsf{P}^{\mathsf{sid}}$ reacts for all other $\hat{c}$ cases: instead of querying $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, \hat{c})$ to get $\hat{pk}$ and generate corresponding e, τ and K, we set $\hat{pw} = \mathsf{c2pw}[\hat{c}]$ and consider two cases: (case 1) if $\hat{pw} = (\mathsf{fullsid}, pw)$ then Game 5 computes K in the same way as in Game 3, except we render the decryption query as retrieval from table $\mathsf{TRIC}_{\hat{pw}}$, which is just a notational change; (case 2) In any other case, Game 5 shortcuts the decryption and key-computation process and outputs a random e, with τ, K as random elements in $\{0,1\}^{\kappa}$. On the other side, $\mathsf{CP}^{\mathsf{sid}}$ computes K as before.

We argue that the change introduced in case 2 is negligible, i.e. if $\mathsf{c2pw}[\hat{c}]$ contains an entry $\hat{pw} \neq (\mathsf{fullsid}, pw)$, including $\hat{pw} = \perp$. The argument is hybrid and changes the view in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ invoked by $\mathcal{Z}$. We consider two sub-cases, (case 2a) where $\hat{c}$ was created via an adversarial encryption query on some key $\hat{pw}$, which does not match the decryption key $(\mathsf{fullsid}, pw)$ that $\mathsf{P}^{\mathsf{sid}}$ would use in Game 3 to decrypt this ciphertext (note that this $\hat{pw}$ is unique because in Game 1 we add an abort if two encryption queries ever create the same ciphertext), and (case 2b) where $\hat{c}$ was not created in any encryption query. In either of these two sub-cases Game 3 would compute $\tau \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e)$, and compute $K \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e, \tau)$ for $(\hat{r}, \hat{pk}) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$ and $(e, K^*) \leftarrow \mathsf{enc}(\hat{pk})$, and since in either case $\hat{c}$ was not inserted in table $\mathsf{TRIC}_{(\mathsf{fullsid}, pw)}$ via an encryption query, this AdvDec query will embed a random $\hat{pk}$ into the decrypted plaintext.

Game 0: real-world interaction

Initialize Cset $= \{\}$ and empty table $\mathsf{TRIC}_{\hat{pw}}$ for all $\hat{pw}$;

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$

if $b = 0$: $(sk, pk) \xleftarrow{r} \mathsf{KEM.kg}$, $c \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, pk)$, save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, pk, c, \bot)$, [output c]

if $b = 1$: save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \bot, \bot, \bot, \bot)$

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one): if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \bot, \bot, \bot, \bot)$:

$(\hat{r}, \hat{pk}) \leftarrow \mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$

$(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e)$

$K \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e, \tau)$

reset $\mathsf{Rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$

[output (e, τ) and $(\mathsf{sid}, \mathsf{P}, K)$]

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one):

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, pk, c, \bot)$:

$K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$

if $\tau = \mathsf{H}(K^*, pk, c, \hat{e})$ then set $K' \leftarrow \mathsf{H}(K^*, pk, c, \hat{e}, \tau)$ and [output $(\mathsf{sid}, \mathsf{P}, K')$]

else [output $K' \xleftarrow{r} \{0,1\}^\kappa$]

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, M)$ (assuming $M \in \mathcal{PK}$): $r \xleftarrow{r} \mathcal{R}$, set $m \leftarrow (r, M)$

If $\exists$ c s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$: return c

else:$c \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{c}$, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, c to Cset, return c

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$: $(m \in \mathcal{R} \times \mathcal{PK}$ and $T \in \mathcal{PK})$: if $\exists$ c s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$: return c

else: $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$

add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$ and c to Cset, return c

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$ (assuming $c \in \mathcal{R} \times \mathcal{PK}$):

if $\exists$ m s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$: return m

else: $m \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{m}$, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, return m

On $\mathcal{A}$'s query x to H:

If $\exists \langle x, y \rangle$ in $\mathsf{T_H}$ output y, else output $y \xleftarrow{r} \{0,1\}^\kappa$ and add $\langle x, y \rangle$ to $\mathsf{T_H}$.

Game 1: randomizing first message

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$

if $b = 0$: set $(sk, pk) \xleftarrow{r} \mathsf{KEM.kg}$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, pk)$, $c \xleftarrow{r} \mathcal{D}$, *abort* if $(m, \cdot) \in \mathsf{TRIC}_{\hat{pw}}$ or $c \in$ Cset, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, pk, c, \bot)$, [output c]

on $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}(\hat{pw}, M)$ or $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:

Before either process adds c to Cset, *abort* if $c \in$ Cset and set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$

Game 2: embedding public key in decryption queries

on query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:

if $\exists m$ s.t. $(m, c) \in \mathsf{TRIC}_{\hat{pw}}$ return m, otherwise set $(sk, pk) \xleftarrow{r} \mathsf{KEM.kg}$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, pk)$, *abort* if $(m, \cdot) \in \mathsf{TRIC}_{\hat{pw}}$, add (m, c) to $\mathsf{TRIC}_{\hat{pw}}$, save $(\mathsf{backdoor}, c, \hat{pw}, sk, pk)$, return (r, pk)

Game 3: delegating key generation to AdvDec

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$

if $b = 0$: $c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in$ Cset, otherwise add c to Cset

query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}(\hat{pw}, c)$, retrieve $(\mathsf{backdoor}, c, \hat{pw}, sk, pk)$

save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, sk, pk, c, \bot)$ and [output c]

Game 456: randomizing second message and keys

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $\mathsf{rec} = (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \bot, \bot, \bot)$:

if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 0, pw, sk, pk, \hat{c}, \bot)$:

pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \leftarrow \{0,1\}^\kappa$ ($pk^* \xleftarrow{r} \mathcal{PK}$, $e \leftarrow c[pk^*]$ is a shortcut for $(e, \cdot) \leftarrow \mathsf{KEM.enc}(pk^*)$)

else: $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$

if $\hat{pw} = (\mathsf{fullsid}, pw)$: retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$

set $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{H}(K^*, pk, \hat{c}, e)$

$K \leftarrow \mathsf{H}(K^*, pk, \hat{c}, e, \tau)$

else: pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$

reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$

[output (e, τ), $(\mathsf{sid}, \mathsf{P}, K)$]

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, pk, c, \bot)$:

if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 1, pw, c, \hat{e}, \hat{\tau}, K)$ then set $K' \leftarrow K$, [output $(\mathsf{sid}, \mathsf{P}, K')$]

else: set $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$

if $\hat{\tau} = \mathsf{H}(K^*, pk, c, e)$ then set $K' \leftarrow \mathsf{H}(K^*, pk, c, e, \hat{\tau})$, and [output $(\mathsf{sid}, \mathsf{P}, K')$]

otherwise [output $K' \xleftarrow{r} \{0,1\}^\kappa$]

Game 7: delaying password usage

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$

if $b = 0$: $c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in$ Cset, otherwise add c to Cset

save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \bot, c, \bot)$ and [output c]

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \bot, \bot, \bot, \bot)$:

if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 0, pw, \bot, \bot, \hat{c}, \bot)$:

pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$

else: $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$

if $\hat{pw} = (\mathsf{fullsid}, pw)$: query $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$, retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{TRIC}_{\hat{pw}}$, set $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e)$, $K \leftarrow \mathsf{H}(K^*, \hat{pk}, \hat{c}, e, \tau)$

else: pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$

reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$

[output (e, τ) and $(\mathsf{sid}, \mathsf{P}, K)$]

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, \bot, \bot, c, \bot)$:

if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 1, pw, c, \hat{e}, \hat{\tau}, K)$ then set $K' \leftarrow K$, and [output $(\mathsf{sid}, \mathsf{P}, K')$]

else if $\exists$ pw s.t. $\exists$ $(\mathsf{backdoor}, c, (\mathsf{fullsid}, pw), sk, pk)$

set $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$

if $\hat{\tau} = \mathsf{H}(K^*, pk, c, \hat{e})$ then set $K' \leftarrow \mathsf{H}(K^*, pk, c, \hat{e}, \hat{\tau})$ and [output $(\mathsf{sid}, \mathsf{P}, K')$]

otherwise [output $K' \xleftarrow{r} \{0,1\}^\kappa$]

Figure 5.17: Game changes for the proof of Theorem 5.3

We make the following changes. First we pick a random K^* instead of via output of $\mathsf{KEM.enc}(pk)$, we argue that an adversary who distinguishes this change with non-negligible advantage implies an attack on the security property of the KEM scheme. Given the KEM security challenge (pk^*, e^*, K^*), the reduction does the following: it guesses an index $i \xleftarrow{r} [1, ..., q_{IC} + q_P]$ of a query to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ using key $(\mathsf{fullsid}, pw)$ and embeds challenge value pk^* into the decrypted plaintext.

If the guess is right and the adversary sends ciphertext $\hat{c}$ used in this i-th query to $\mathsf{P}^{\mathsf{sid}}$, the reduction embeds the (e^*, K^*) challenge value into the KEM.enc output by $\mathsf{P}^{\mathsf{sid}}$. (If the guess is not right the reduction aborts.) If the guess is right, the reduction reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 3 if K^* is the real key corresponding to KEM instance (pk^*, e^*, K^*), and it reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 5 if K^* is random. Since the right guess occurs with probability $1/(q_{IC}+q_P)$ and the identity of the index i does not affect the view the reduction produces before the abort, and the argument goes by a hybrid over all honest party sessions, this change is upper-bounded by $q_P/(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.sec}}$.

Then we pick τ at random instead of via H, with an abort if τ has been output by H before. We also change e to be directly generated via a random public key pk^* picked by SIM, which is only a syntactic change because as mentioned in previous part of this game, in case 2 AdvDec query will always embed a random pk into the decrypted plaintext, and generate e based on this pk. Now that $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ is not used in response to $\hat{c}$, we further remove query to $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$.

Note that process on $\mathsf{CP}^{\mathsf{sid}}$ remains unchanged, where τ checking always fails and $\mathsf{CP}^{\mathsf{sid}}$ always output a random session key as in the previous game. We conclude that:

$$|P_6 - P_5| \leq q_P/(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.sec}} + \frac{q_H}{2^\kappa} \tag{5.14}$$

GAME 7 (delaying password usage): In this game we delay using the password pw of session $\mathsf{P}^{\mathsf{sid}}$ to decrypt its outgoing message c to the moment when $\mathsf{P}^{\mathsf{sid}}$ receives an incoming message $(\hat{e}, \hat{\tau})$ in the case of an active attack, where $\mathcal{A}$ made a decryption query on c using correct password. In this case $\mathsf{P}^{\mathsf{sid}}$ will go through the list of backdoor records based on pw and c to retrieve sk for KEM decapsulation, whereas Game 5 retrieves sk from $\mathsf{P}^{\mathsf{sid}}$'s record. This is just a notational change. We also change how $\mathsf{CP}^{\mathsf{sid}}$ reacts to an incoming message $\hat{c}$, and we perform the decryption query only in the case adversary created $\hat{c}$ via encryption under correct key $(\mathsf{fullsid}, pw)$. Since Game 5 does not use the decrypted value (r, pk) and the associated trapdoor sk until the exact same case occurs, postponing this decryption does not matter as long as item $(*, c)$ is not written into table $\mathsf{TRIC}_{(\mathsf{fullsid}, pw)}$ via an encryption query. However, the latter cannot happen in Game 5 because each NewSession and each encryption queries generate disjoint ciphertexts (a collision in the ciphertexts created by any of these queries leads to an abort). Both cases imply that $P_6 = P_7$.

GAME 8 (ideal-world game implied by $\mathcal{F}_{\mathsf{pwKE}}$ and SIM): This is the ideal-world game induced by functionality $\mathcal{F}_{\mathsf{pwKE}}$ interacting with simulator SIM of Figure 5.16. Since Game 7 matches the ideal-world interaction of Game 8 exactly we conclude that $P_7 = P_8$, which completes the proof.

□

5.5 Applications of HIC to asymmetric PAKE

Gu et al. [74] proposed an asymmetric PAKE protocol called KHAPE which is a generic compiler from any UC *key-hiding* Authenticated Key Exchange (AKE), using an Ideal Cipher on the domain formed by (private, public) key pairs of the AKE.

Here we claim that KHAPE remains a UC aPAKE[12] if the Ideal Cipher used to encrypt the private and public AKE keys in protocol KHAPE is replaced by a Randomized Ideal Cipher . The benefit of replacing IC^* implementation of the ideal cipher on a group in [74] with a RIC is that, as we show with the m2F construction of RIC, the latter can be implemented over any group which admits an RO-indifferentiable random oracle hash onto a group, requires only one such hash to both encrypt and decrypt, and it has bandwidth overhead of 2κ bits. By contrast, the IC^* implementations of an ideal cipher on a group suggested in [74] work only for restricted elliptic curve groups and/or require more bandwidth and more computation in encryption and decryption. The same change can also benefit protocol OKAPE[68], which improves the round efficiency of KHAPE, and the change should be done similarly to KHAPE.

We show the KHAPE protocol using Randomized Ideal Cipher for password-encryption of keys in Figure 5.18. Intuitively Randomized Ideal Cipher works because: in KHAPE the attacker can attack client by sending an arbitrary ciphertext of his choice, but with the credential encryption implemented using an ideal cipher, the ciphertext commits the attacker to only one choice of key/password, for which he can decide the plaintext. And for all other keys the decrypted plaintext will be random, i.e. there are two requirements: (1) ciphertext $c = \mathsf{Enc}(k, m)$ is an encryption of some unique (k, m). RIC satisfies since for every (k, m) RIC.Enc and RIC.AdvEnc outputs $c = (s, T)$ which has no collisions on s part; (2) $\mathsf{Dec}(k', c)$ for $k' \neq k$ outputs random M, which is defined as in RIC.AdvDec and RIC.Dec.

For reference, for AKE functionality $\mathcal{F}_{\mathsf{khAKE}}$ see e.g., [74], and for aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ see Figure 2.4 e.g., [67].

Theorem 5.4. *Protocol* KHAPE *of [74] realizes the UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *if the AKE protocol realizes the Key-Hiding AKE functionality* $\mathcal{F}_{\mathsf{khAKE}}$ *assuming that* kdf *is a secure PRF and* RIC *is a randomized ideal cipherover message space of private and public key pairs in AKE.*

[12] The UC asymmetric PAKE functionality, adapted to the case of explicit C-to-S authentication implemented by protocol KHAPE, is shown in Section 2.3.3.

We note that Freitas et al. [68] showed a UC aPAKE which improves upon protocol KHAPE of [74] in round complexity. The aPAKE of [68] relies on IC in a similar way as protocol KHAPE, and the proof therein should also generalize to the case when IC is replaced by HIC.

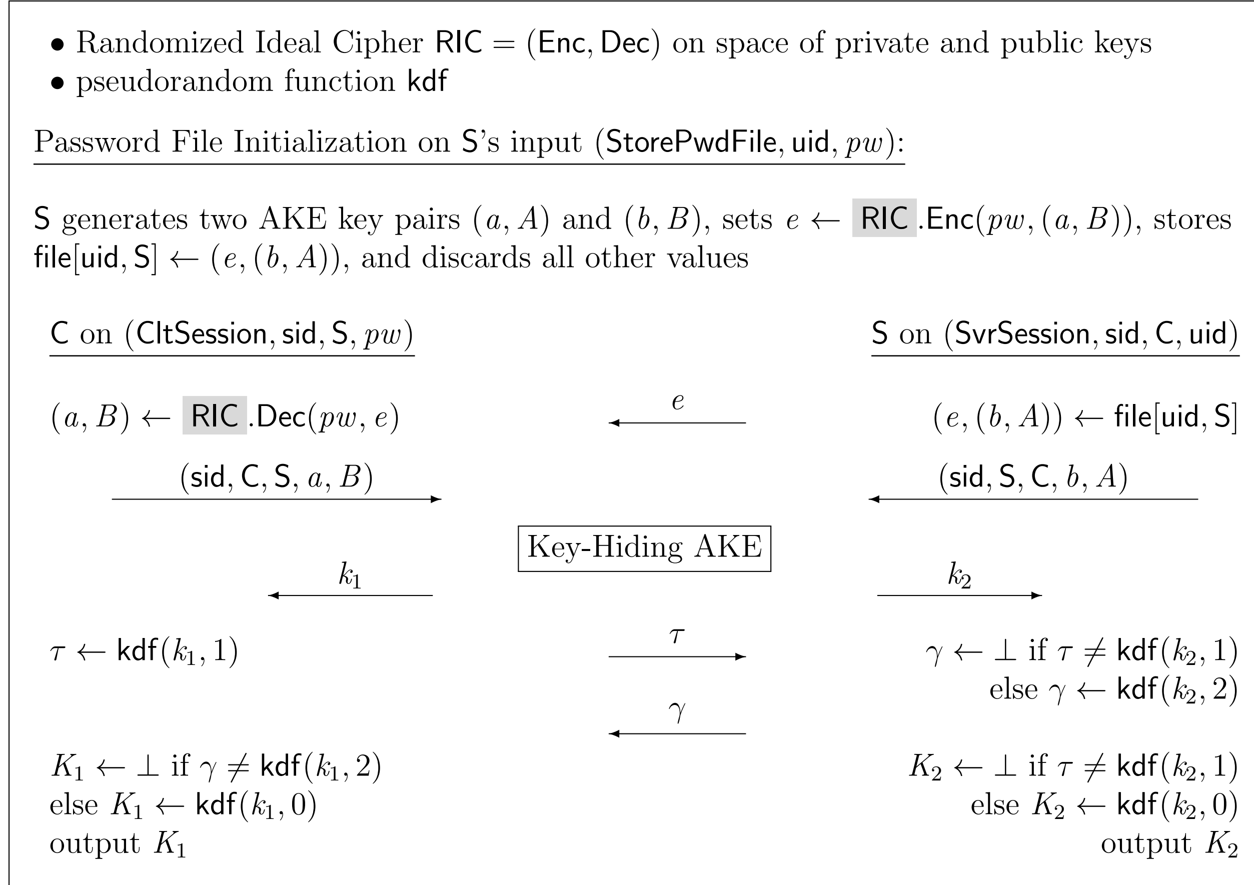


Figure 5.18: protocol KHAPE using Randomized Ideal Cipher (changes from [74] marked so)

Proof. We describe how the security proof for KHAPE should be adapted to the case using RIC. We specify how we deal with the RIC-specific differences when they occur and mark them in gray. We show that the environment's view of the *real-world* security game, denoted Game 0, i.e. an interaction between the real-world adversary and honest parties who follow protocol KHAPE, is indistinguishable from the environment's view of the *ideal-world* game, denoted Game 8, i.e. an interaction between simulator SIM[13] of Figures 5.20 and functionality $\mathcal{F}_{\mathsf{aPAKE}}$2.4. As before, we use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with

[13]here we only attach part of the simulator since the rest, i.e. the "Respodning to AKE messages" part, is same as in [74]

Initialize empty table TRIC. Notation $\mathsf{TRIC}_{\hat{pw}}.X'$, $\mathsf{TRIC}_{\hat{pw}}.Y$ and $\mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$ as in Fig. 5.20. Code below assumes queries $\mathsf{Enc}, \mathsf{AdvEnc}, \mathsf{AdvDec}, \mathsf{Dec}$ are *new*.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Generate keys (a, A), (b, B), set $e_{\mathsf{S}}^{\mathsf{uid}} \leftarrow \mathsf{Enc}(pw_{\mathsf{S}}^{\mathsf{uid}}, (a, B))$, and $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$
- On (pw, x) to Enc: pick $r \xleftarrow{r} \mathcal{R}$, set $x' \leftarrow (r, x)$, output $y \xleftarrow{r} Y \setminus \mathsf{TRIC}_{\hat{pw}}.Y$, add (pw, x', y) to TRIC
- On (pw, x', T) to AdvEnc: pick $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, $y \leftarrow (s, T)$, add (pw, x', y) to TRIC, and output y
- On (pw, y) to AdvDec: Output $x' \xleftarrow{r} X' \setminus \mathsf{TRIC}_{\hat{pw}}.X'$, add (pw, x', y) to TRIC
- On (pw, y) to Dec: Query $(r, x) \leftarrow \mathsf{AdvDec}(pw, y)$, output x
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Set $(e_{\mathsf{S}}^{\mathsf{uid}}, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, send $e_{\mathsf{S}}^{\mathsf{uid}}$ and start AKE session $\mathsf{S}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)$, set k_2 to $\mathsf{S}^{\mathsf{sid}}$ output;
  If $\mathcal{Z}$ sends $\tau' = \mathsf{kdf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set K_2, γ as $\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2)$, else as $\perp, \perp$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message e' to C: Set $(a, B) \leftarrow (\mathsf{Dec}(pw, e'))$, and start AKE session $\mathsf{C}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$, set k_1 to $\mathsf{C}^{\mathsf{sid}}$ output, send $\tau = \mathsf{kdf}(k_1, 1)$ to $\mathcal{Z}$;
  If $\mathcal{Z}$ sends $\gamma' = \mathsf{kdf}(k_1, 2)$ to $\mathsf{C}^{\mathsf{sid}}$, set $K_1 = \mathsf{kdf}(k_1, 0)$ else $K_1 = \perp$

Figure 5.19: Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE

Game i, and the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G8}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let q_{pw}, q_{RIC}, and q_{ses} be the upper-bounds on the number of resp. password files, RIC queries, and online S or C aPAKE sessions. Let $\epsilon_{\text{kdf}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$ and $\epsilon_{\text{ake}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$ be the advantages of an environment who uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{RIC}} + q_{\text{ses}} + q_{\mathsf{pw}})$ exponentiations in $\mathbb{G}$ in resp. breaking the PRF security of kdf, and in distinguishing between the real-world AKE protocol and its ideal-world emulation of $\mathsf{SIM}_{\mathsf{AKE}}$ interacting with $\mathcal{F}_{\mathsf{khAKE}}$. Let $X' = Y = \mathcal{R} \times \mathcal{G}$ be the domain and range of the Randomized Ideal Cipher RIC used, let X be the domain of (private,public) keys in AKE(e.g. for both 3DH and HMQV we have $X = \mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p). Whereas [74] defined a mapping from groups to bitstrings and used a "bitstring" IC on the result, here we directly show a (randomized) IC on groups, precisely so

that it can be used directly for public key systems where public keys live in groups[14], which is the case for all public keys we give as our examples (either DH-based or Lattice-based).

GAME 0 (real world): This is the interaction, shown in Figure 6.6, of environment $\mathcal{Z}$ with the real-world protocol KHAPE, except that the symmetric encryption scheme is idealized as a Randomized Ideal Cipher oracle. (Technically, this is a hybrid world where each party has access to the Randomized Ideal Cipher functionality $\mathcal{F}_{\mathsf{RIC}}$.)

GAME 1 (embedding random keys in $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvDec}$ outputs): We modify processing of $\mathcal{Z}$'s query (pw, y) to AdvDec[15] for any $y \notin \mathsf{TRIC}_{\hat{pw}}.Y$, i.e. y for which $\mathsf{AdvDec}(pw, y)$ has not been yet defined. On such query Game 1 pick a random r, generates fresh key pairs (a, A) and (b, B), sets $x' \leftarrow (r, (a, B))$, and if $x' \notin \mathsf{TRIC}_{\hat{pw}}.X'$ then it sets $\mathsf{AdvDec}(pw, y) \leftarrow x'$. If $x' \in \mathsf{TRIC}_{\hat{pw}}.X'$, i.e. x' is already generated by $\mathsf{AdvEnc}(pw, \cdot, \cdot)$ or $\mathsf{Enc}(pw, \cdot)$, Game 1 aborts. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then the game also sets $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (r, A, B)$.

The divergence this game introduces is due to the probability $(q_{\mathsf{RIC}})^2/2^n$ of ever encountering an abort [16], which leads to $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (q_{\mathsf{RIC}})^2/2^n$.

GAME 2 (random $e_{\mathsf{S}}^{\mathsf{uid}}$ in the password file): We change $\mathsf{StorePwdFile}$ processing by picking ciphertext $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random element in $\{0,1\}^n \times \mathbb{G}$ instead of via query to Enc, then we pick two key pairs (a, A), (b, B), pick a random r and define $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r, A, B)$, set $x' \leftarrow (r, (a, B))$. If $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{TRIC}_{\hat{pw}}.Y$ for *any* pw, not necessarily $pw_{\mathsf{S}}^{\mathsf{uid}}$, the game aborts. The game also aborts if $x' \in \mathsf{TRIC}_{\hat{pw}}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$. Otherwise the game sets $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow x'$ and $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r, A, B)$. The divergence this game introduces is due to the probability of abort occuring in either case, which leads to $|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq 2q_{\mathsf{pw}}q_{\mathsf{RIC}}/2^n$.

GAME 3 (abort on ambiguous ciphertexts): In[74] to eliminate the possibility of ambiguous

[14] the secret keys in our cases are either $\mathbb{Z}_p$ elements or bitstrings which are in groups
[15] all the Enc,AdvEnc,AdvDec notation refers to oracles defined by $\mathcal{F}_{\mathsf{RIC}}$
[16] the probability of collission comes from the n-bit string $r$is is at most $(q_{\mathsf{RIC}})^2/2^n$

Initialization

Initialize simulator $\mathsf{SIM}_{\mathsf{AKE}}$, an empty table TRIC, empty lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$

Notation: $\mathsf{TRIC}_{\hat{pw}}.X' = \{x' \mid (pw, x', \cdot) \in \mathsf{TRIC}\}$, $\mathsf{TRIC}_{\hat{pw}}.Y = \{y \mid (pw, \cdot, y) \in \mathsf{TRIC}\}$, $\mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T] = \{s \in \mathcal{R} \mid (\cdot, (s, T)) \notin \mathsf{TRIC}_{\hat{pw}}\}$.

Convention: First call to $\mathsf{SvrSession}$ or $\mathsf{StealPwdFile}$ for $(\mathsf{S}, \mathsf{uid})$ sets $e_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} Y$. W.l.o.g. we assume $\mathcal{A}$ uses AdvDec interface to implement $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Dec}$ queries.

Randomized Ideal Cipher queries

- On query (Enc, pw, x) to $\mathcal{F}_{\mathsf{RIC}}$, send back y if $(pw, (\cdot, x), y) \in \mathsf{TRIC}$, else pick $r \xleftarrow{r} \mathcal{R}$, $y \xleftarrow{r} Y \setminus \mathsf{TRIC}_{\hat{pw}}.Y$, add $(pw, (r, x), y)$ to TRIC, send back y
- On query $(\mathsf{AdvEnc}, pw, x', T)$ to $\mathcal{F}_{\mathsf{RIC}}$, send back y if $(pw, x', y) \in \mathsf{TRIC}$, else pick $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, set $y \leftarrow (s, T)$, add (pw, x', y) to TRIC, send back y
- On query (AdvDec, pw, y) to $\mathcal{F}_{\mathsf{RIC}}$, send back x' if $(pw, x', y) \in \mathsf{TRIC}$, else do:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \xleftarrow{r} X' \setminus \mathsf{TRIC}_{\hat{pw}}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and:
     (a) If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" then set $(r, A, B) \leftarrow (r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     (b) Otherwise pick $r \xleftarrow{r} \mathcal{R}$, initialize keys A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}

     Set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (r, A, B)$, send query $(\mathsf{Compromise}, A)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define a as $\mathsf{SIM}_{\mathsf{AKE}}$'s response, add A to CPK, set $x' \leftarrow (r, (a, B))$

  In either case add (pw, x', y) to TRIC and send back x'

Stealing Password Data

On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$. If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "no password file," pass it to $\mathcal{A}$, otherwise declare $(\mathsf{S}, \mathsf{uid})$ compromised and:

1. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns no value then pick $r \xleftarrow{r} \mathcal{R}$, initialize keys A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}
2. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns pw then set $(r, A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw)$

Send $(\mathsf{Compromise}, B)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define b as $\mathsf{SIM}_{\mathsf{AKE}}$'s response, add B to CPK, set $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r, A, B)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$ to $\mathcal{A}$.

Starting AKE sessions

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$, initialize random function $R_{\mathsf{S}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $e_{\mathsf{S}}^{\mathsf{uid}}$ to $\mathcal{A}$ as a message from $\mathsf{S}^{\mathsf{sid}}$, and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$ to $\mathsf{SIM}_{\mathsf{AKE}}$.

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$ and message e' sent by $\mathcal{A}$ to $\mathsf{C}^{\mathsf{sid}}$, initialize random function $R_{\mathsf{C}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, and:

1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ check if e' was output by $\mathcal{F}_{\mathsf{RIC}}.\mathsf{Enc}$ on some (pw, x) or $\mathcal{F}_{\mathsf{RIC}}.\mathsf{AdvEnc}$ on some $(pw, (r, x))$, and:
   (a) If there is no such query then send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
   (b) Otherwise define (pw, x) (resp. $(pw, (r, x))$) as the first such query(abort others) which outputted e', send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, and:
      i. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "wrong guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
      ii. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess" then set $(a, B) \leftarrow x$ and run the AKE protocol on behalf of $\mathsf{C}^{\mathsf{sid}}$ on inputs $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$; When $\mathsf{C}^{\mathsf{sid}}$ terminates with key k then send $\tau \leftarrow \mathsf{kdf}(k, 1)$ to $\mathcal{A}$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$

Figure 5.20: Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$

ciphertexts we introduce an abort if $\mathsf{IC.Enc}$ oracle picks the same ciphertext for any two queries containing pair (pw_1, x_1) and (pw_2, x_2). Now this ambiguous case is already considered and avoided in definition of Enc and AdvEnc in $\mathcal{F}_{\mathsf{RIC}}$. so we have $\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$.

Taking stock of the game. Let us review how Game 3 operates: The initialization of password file $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ on password $pw_{\mathsf{S}}^{\mathsf{uid}}$ picks a random r and fresh keys (a, A), (b, B), keeps them as $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) = (r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) = (r, A, B)$, picks $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random string, and programs $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ to $(r, (a, B))$. Oracle AdvDec on inputs (pw', y) for which decryption is undefined, picks some random r' and fresh key pairs (a', A') and (b', B'), and programs $\mathsf{AdvDec}(pw', y)$ to $(r', (a', B'))$. In addition, if $y = e_{\mathsf{S}}^{\mathsf{uid}}$ then it assigns $pk_{\mathsf{S}}^{\mathsf{uid}}(pw') \leftarrow (r', (a', B'))$. Finally, encryption is now unambiguous, i.e. every ciphertext e can be output by Enc or AdvEnc on only one pair (pw, x').

This is already very close to how simulator SIM operates as well. The crucial difference between the ideal-world interaction and Game 3, is that in Game 3, $r_{\mathsf{S}}^{\mathsf{uid}}$ and keys $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ are generated at the time of password file initialization, and $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set to $(r_{\mathsf{S}}^{\mathsf{uid}}, (a_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}))$ at the same time. In the ideal-world game these values are undefined until password compromise, and $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set only after offline dictionary attack succeeds in finding $pw_{\mathsf{S}}^{\mathsf{uid}}$. This delayed generation of the keys in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ is possible because AKE sessions which S and C run on these keys can be simulated without knowledge of these keys, a key-hiding AKE functionality allows precisely for such simulation, as we show next. Delayed r generation is also okay because it's not used in AKE sessions.

Game 4 (Using $\mathsf{SIM}_{\mathsf{AKE}}$ for AKE's on honestly-generated keys): In Game 4 we modify Game 3 by replacing all honest parties that run AKE instances on keys A, B generated either in password file initialization or by oracle AdvDec, with a simulation of these AKE instances via simulator $\mathsf{SIM}_{\mathsf{AKE}}$. For notational brevity we say that query (pw, x) to Enc(resp. (pw, x', T) to AdvEnc) or (pw, y) to AdvDec are new$^{(!)}$ as a shortcut for saying that table TRIC includes no prior tuple corresponding to these inputs. If such tuple exists then Enc,

Initialize simulator $\mathsf{SIM_{AKE}}$, empty table TRIC and $\mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, and lists $CPK, PK_\mathsf{C}, PK_\mathsf{S}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid})$ to S: Pick $e_\mathsf{S}^\mathsf{uid} \xleftarrow{r} Y$, mark $pw_\mathsf{S}^\mathsf{uid}$ as fresh
- On new$^{(!)}$ (pw, x) to Enc: Pick $r \xleftarrow{r} \mathcal{R}$, set $x' \leftarrow (r, x)$, output $y \xleftarrow{r} Y \setminus \mathsf{TRIC}_{\hat{pw}}.Y$, add (pw, x', y) to TRIC
- On new$^{(!)}$ (pw, x', T) to AdvEnc: Pick $s \xleftarrow{r} \mathsf{TRIC}_{\hat{pw}}.\mathsf{s}[T]$, $y \leftarrow (s, T)$, add (pw, x', y) to TRIC, and output y
- On new$^{(!)}$ (pw, y) to AdvDec, do cases below then add (pw, x', y) to TRIC and output x':
  1. If $y \neq e_\mathsf{S}^\mathsf{uid}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \xleftarrow{r} X' \setminus \mathsf{TRIC}_{\hat{pw}}.X'$
  2. If $y = e_\mathsf{S}^\mathsf{uid}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_\mathsf{S}^\mathsf{uid}$ is fresh or $pw \neq pw_\mathsf{S}^\mathsf{uid}$ then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, pick $r \xleftarrow{r} \mathcal{R}$, initialize A and B via Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_C and B to PK_S
     (b) If $pw_\mathsf{S}^\mathsf{uid}$ is $\mathsf{compromised}$&$pw = pw_\mathsf{S}^\mathsf{uid}$, set $(r, A, B) \leftarrow (r_\mathsf{S}^\mathsf{uid}, A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid})$

     In both cases (a) and (b), set $pk_\mathsf{S}^\mathsf{uid}(pw) \leftarrow (r, A, B)$, define a as $\mathsf{SIM_{AKE}}$'s response to $(\mathsf{Compromise}, A)$, add A to CPK, and set $x' \leftarrow (r, (a, B))$
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: mark $pw_\mathsf{S}^\mathsf{uid}$ $\mathsf{compromised}$ and: If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid} \rangle$ then set $(r, A, B) \leftarrow pk_\mathsf{S}^\mathsf{uid}(pw_\mathsf{S}^\mathsf{uid})$; Else pick $r \xleftarrow{r} \mathcal{R}$, initialize A, B via Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_C and B to PK_S; In either case, set $(r_\mathsf{S}^\mathsf{uid}, A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid}) \leftarrow (r, A, B)$, define b as $\mathsf{SIM_{AKE}}$'s response to $(\mathsf{Compromise}, B)$, add B to CPK, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_\mathsf{S}^\mathsf{uid}, b, A)$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_\mathsf{S}^\mathsf{sid}$, set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{hbc}$, output $e_\mathsf{S}^\mathsf{uid}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$ to $\mathsf{SIM_{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_\mathsf{C}^\mathsf{sid}$ and:
  1. If $e' = e_\mathsf{S}^\mathsf{uid}$ then: (1) set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{hbc}_\mathsf{S}^\mathsf{uid}$ if $pw = pw_\mathsf{S}^\mathsf{uid}$, otherwise set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{rnd}$; (2) send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM_{AKE}}$
  2. If $e' \neq e_\mathsf{S}^\mathsf{uid}$ then:
     (a) If e' was not output by Enc or AdvEnc or it was output on $(pw', \cdot)$ for $pw' \neq pw$, then set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM_{AKE}}$
     (b) If e' was output by Enc on (pw, x) or AdvEnc on $(pw, (\cdot, x), \cdot)$ then set $(a, B) \leftarrow x$, run C^sid of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If C^sid terminates with k, output $\tau \leftarrow \mathsf{kdf}(k, 1)$ and $K_1 \leftarrow \mathsf{kdf}(k, 0)$

Responding to AKE messages:

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}_\mathsf{S}^\mathsf{uid}$ then $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{act}_\mathsf{S}^\mathsf{uid}$ if $pw_\mathsf{S}^\mathsf{uid}$ is $\mathsf{compromised}$, otherwise $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{rnd}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{act}_\mathsf{S}^\mathsf{uid}$ set $k_1 \leftarrow R_\mathsf{C}^\mathsf{sid}(A_\mathsf{S}^\mathsf{uid}, B_\mathsf{S}^\mathsf{uid}, \alpha)$, output $\tau \leftarrow \mathsf{kdf}(k_1, 1)$
  2. Otherwise output $\tau \xleftarrow{r} \{0,1\}^\kappa$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to S^sid:
  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}$ and $\tau' = \mathsf{kdf}(k_2, 1)$ for $k_2 = R_\mathsf{S}^\mathsf{sid}(B, A, \alpha)$ where $(\cdot, (A, B)) = pk_\mathsf{S}^\mathsf{uid}(pw_\mathsf{S}^\mathsf{uid})$, then output $(K_2, \gamma) \leftarrow (\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2))$
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$ and τ' was generated by C^sid where $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}_\mathsf{S}^\mathsf{uid}$, then output $K_2 \xleftarrow{r} \{0,1\}^\kappa$ and $\gamma \xleftarrow{r} \{0,1\}^\kappa$
  3. In all other cases output $(K_2, \gamma) \leftarrow (\bot, \bot)$
- On γ' to C^sid:
  1. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{act}_\mathsf{S}^\mathsf{uid}$ and $\gamma' = \mathsf{kdf}(k_1, 2)$, output $K_1 \leftarrow \mathsf{kdf}(k_1, 0))$
  2. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}_\mathsf{S}^\mathsf{uid}$ and γ' was generated by S^sid for S^sid s.t. $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$, output K_1 equal to the key K_2 output by S^sid
  3. In all other cases output $K_1 \leftarrow \bot$
- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_\mathsf{P}^\mathsf{sid}(pk, pk', \alpha)$ if $pk \in PK_\mathsf{P}$, $pk' \in CPK$

Figure 5.21: KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 8)

AdvEnc and AdvDec oracles use the retrieved (key,input,output) tuple to answer the according query. We also omit the possibilities of the game aborts, because such aborts happen only with negligible probability. These aborts occur in three places, all marked $^{(*)}$: (1) When $e_{\mathsf{S}}^{\mathsf{uid}}$ is chosen in $\mathsf{StorePwdFile}$ the game aborts if $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{TRIC}_{\hat{pw}}.Y$ for any pw (not necessarily $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$); (2) When x' is then set as $x' \leftarrow (r, (a, B))$, the game aborts if $x' \in \mathsf{TRIC}_{\hat{pw}}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$; (3) When $x' \leftarrow (r, (a, B))$ is set in AdvDec query (pw, y) the game aborts also if $x' \in \mathsf{TRIC}_{\hat{pw}}.X'$.

Game 4 operates like Game 3, except that it outsources AKE key generation in $\mathsf{StorePwdFile}$ and AdvDec to $\mathsf{SIM}_{\mathsf{AKE}}$, and whenever $\mathsf{S}^{\mathsf{sid}}$ or $\mathsf{C}^{\mathsf{sid}}$ runs AKE on such keys these executions are outsourced to $\mathsf{SIM}_{\mathsf{AKE}}$, while the game emulates what $\mathcal{F}_{\mathsf{khAKE}}$ would do in response to $\mathsf{SIM}_{\mathsf{AKE}}$'s actions. In particular, Game 4 initializes random function $R_{\mathsf{P}}^{\mathsf{sid}}$ for every AKE session $\mathsf{P}^{\mathsf{sid}}$ invoked by emulated $\mathcal{F}_{\mathsf{khAKE}}$. Whenever C and S run an AKE instance under keys generated by AKE key generation the game, playing $\mathcal{F}_{\mathsf{khAKE}}$, triggers $\mathsf{SIM}_{\mathsf{AKE}}$ with messages resp. $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$. When $\mathsf{SIM}_{\mathsf{AKE}}$ translates the real-world adversary's behavior into $\mathsf{Interfere}$ actions on these sessions, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by marking these sessions as actively attacked. If $\mathsf{SIM}_{\mathsf{AKE}}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ on activey attacked session, its output key k is set to $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the keys this session runs under, which are $(B_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}})$ for S, and keys (A, B) defined by $\mathsf{AdvDec}(pw, e')$ for C. The game must also emulate $\mathsf{ComputeKey}$ interface of $\mathcal{F}_{\mathsf{khAKE}}$ and let $\mathsf{SIM}_{\mathsf{AKE}}$ evaluate $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ for any $pk \in PK_{\mathsf{P}}$ and any $pk' \in CPK$. (Note that all sessions emulated by $\mathsf{SIM}_{\mathsf{AKE}}$ run on public keys pk' which are created by the Init interface.) Set PK_{S} contains only one key, $B_{\mathsf{S}}^{\mathsf{uid}}$, while set PK_{C} contains $A_{\mathsf{S}}^{\mathsf{uid}}$ and all keys A' created by AdvDec queries. Set CPK consists of $A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}$, because these were compromised in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ initialization, which used the corresponding private keys, and all client-side keys A' generated in AdvDec queries, because each AdvDec query creates and immediately compromises key A', since it needs to embed the corresponding private key a' into AdvDec output. Finally, if $\mathsf{SIM}_{\mathsf{AKE}}$ sends NewKey on non-attacked session, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by issuing random

keys to such sessions except if $\mathsf{C}^{\mathsf{sid}}$ runs under key pair $(A', B') = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, which matches the key pair used by $\mathsf{S}^{\mathsf{sid}}$, in which case the game copies the key output by the session which terminates first into the key output by the session which terminates second. The rest of the code is as in Game 3: C uses its key k_1 to compute authenticator $\tau = \mathsf{kdf}(k_1, 1)$ and its local output $K_1 = \mathsf{kdf}(k_1, 0)$, while S uses its key k_2 to verify the incoming authenticator τ' and outputs $K_2 = \mathsf{kdf}(k_2, 0)$ if $\tau' = \mathsf{kdf}(k_2, 1)$ and $K_2 = \perp$ otherwise.

The one case where a party might not run AKE on keys generated via a call to $\mathsf{SIM}_{\mathsf{AKE}}$ is client session C which receives e' which was output by $\mathsf{Enc}(pw, x)$ or $\mathsf{AdvEnc}(pw, (\cdot, x), \cdot)$ for some x and pw matching the password input to $\mathsf{C}^{\mathsf{sid}}$. In this case $\mathsf{C}^{\mathsf{sid}}$ runs AKE on $(a, B) = x$, and since wlog these keys are chosen by the adversary and not by $\mathsf{SIM}_{\mathsf{AKE}}$, we cannot outsource that execution to $\mathsf{SIM}_{\mathsf{AKE}}$. As we said above, functionality $\mathcal{F}_{\mathsf{khAKE}}$ does not admit honest parties running AKE on arbitrary private keys a, hence $\mathsf{SIM}_{\mathsf{AKE}}$ does not have an interface to simulate such executions. In Game 4 such AKE instances are executed as in Game 3.

Since Game 4 and Game 3 are identical except for replacing real-world AKE executions with the game emulating functionality $\mathcal{F}_{\mathsf{khAKE}}$ interacting with $\mathsf{SIM}_{\mathsf{AKE}}$, it follows that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon_{\mathrm{ake}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$

GAME 5 (delay $r_{\mathsf{S}}^{\mathsf{uid}}$,$A_{\mathsf{S}}^{\mathsf{uid}}$, $B_{\mathsf{S}}^{\mathsf{uid}}$ generation until password compromise): In Game 4, $r_{\mathsf{S}}^{\mathsf{uid}}$ and keys $A_{\mathsf{S}}^{\mathsf{uid}}$, $B_{\mathsf{S}}^{\mathsf{uid}}$ are initialized and compromised in $\mathsf{StorePwdFile}$, in Game 5 we postpone these steps until password compromise. This change can be done in several steps.

Denote first step as Game 5(a), we remove compromising $B_{\mathsf{S}}^{\mathsf{uid}}$, adding it to CPK and setting $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$. $\mathcal{Z}$ cannot notice this change because in Game 4, only $\mathsf{StealPwdFile}$ will need $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$, and compromising $B_{\mathsf{S}}^{\mathsf{uid}}$ to get $b_{\mathsf{S}}^{\mathsf{uid}}$ is not needed anywhere else except when generating $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$.

In Game 5(b) we make a change in AdvDec, that if $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ then set $x' \xleftarrow{\mathrm{r}} X' \setminus \mathsf{TRIC}_{\hat{pw}}.X'$,

while in Game 4 we set $x' \leftarrow (r, (a, B))$ for randomly initialized $(r, (a, B))$, with restriction that this x' hasn't been set before. This is just a notational change.

Then in Game 5(c) we remove compromising $A_{\mathsf{S}}^{\mathsf{uid}}$, adding it to CPK, setting x' and adding $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ to TRIC in $\mathsf{StorePwdFile}$, and delay them to new$^{(!)}$ (pw, y) to AdvDec. After this change, in $\mathsf{StorePwdFile}$ we now only initialize $r_{\mathsf{S}}^{\mathsf{uid}}$ and $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, add them to PK and pick $e_{\mathsf{S}}^{\mathsf{uid}}$. Since $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ is no longer added to TRIC in $\mathsf{StorePwdFile}$, query $(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is now new$^{(!)}$ to AdvDec, and we add that in this case AdvDec responds by retrieving $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, compromising $A_{\mathsf{S}}^{\mathsf{uid}}$, setting corresponding x' and adding $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ to TRIC. For any other queries, AdvDec reacts same as in Game 5(b). Game 5(c) and Game 5(b) is identical since we only postpone executing those steps removed from $\mathsf{StorePwdFile}$.

In Game 5(d) we further remove usage of $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ when responding to AKE messages, except for input to $R_{\mathsf{P}}^{\mathsf{sid}}$ in actively attacked sessions. We change $\mathsf{hbc}(A, B)$ in Game 5(c) to $\mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ if $(A, B) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, and rnd otherwise. Similarly we change $\mathsf{act}(A, B)$ in Game 5(c) to $\mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ if $(A, B) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, which corresponds to active attack, otherwise set to rnd and derive corresponding k_1 from random element of $\{0, 1\}^{\kappa}$ instead of $R_{\mathsf{C}}^{\mathsf{sid}}(A, B, \alpha)$, from randomness of $R_{\mathsf{C}}^{\mathsf{sid}}$ this change makes indistinguishable difference to $\mathcal{Z}$. Since these are only notational changes and $\mathcal{Z}$ cannot notice them, Game 5(d) and Game 5(c) are identical to $\mathcal{Z}$.

Finally, in Game 5(e) we remove steps of picking $r_{\mathsf{S}}^{\mathsf{uid}}$ and initializing $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via $\mathsf{SIM}_{\mathsf{AKE}}$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$ or $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, depending on which happens first. In order to set $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ only after $\mathcal{A}$ finds $pw_{\mathsf{S}}^{\mathsf{uid}}$ via successful offline dictionary attack, we first mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ fresh in $\mathsf{StorePwdFile}$, and mark it $\mathsf{compromised}$ anytime $\mathcal{A}$ runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$.

If $\mathcal{A}$ first runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, we pick $r_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \mathcal{R}$, initialize $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add $A_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{C} and $B_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{S}, and later upon query $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is already marked $\mathsf{compromised}$, we simply retrieve $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, then compromise $A_{\mathsf{S}}^{\mathsf{uid}}$ and set x' as in Game 5(d). In the other case, if $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ runs first, which means at this moment $pw_{\mathsf{S}}^{\mathsf{uid}}$ must be fresh, we treat it same way as before, and just like any other

$pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$, where we pick $r_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \mathcal{R}$, init $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via $\mathsf{SIM}_{\mathsf{AKE}}$, add them to PK and save $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ into $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ for future retrieval. We also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, and later if $\mathcal{A}$ runs $\mathsf{StealPwdFile}$ and there exists record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, then just directly retrieve $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ from $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ and skip initialization. In addition we also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ upon query $\mathsf{AdvDec}(pw, e_{\mathsf{S}}^{\mathsf{uid}})$ even if $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. Game 5(e) is identical to Game 5(d) since we only postpone $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ initialization. Thus we conclude: $\mathsf{G5} = \mathsf{G4}$

Game 6 (replace kdf output with random string in passive sessions): In Game 5, in passive sessions, i.e. any sessions except actively attacked sessions, τ, γ are all derived from kdf of k_1 or k_2. In Game 6 in these sessions we remove usage of kdf and directly assign random elements of $\{0,1\}^\kappa$ to these values. Also we replace verifying τ', γ' via checking $\tau' = \mathsf{kdf}(k_2, 1), \gamma' = \mathsf{kdf}(k_1, 2)$ with checking whether they're generated by corresponding hbc parties, since these two checking methods are actually equal. In addition, we further remove usage of k_1 and k_2 in passive sessions, and instead set $K_2 \leftarrow \{0,1\}^\kappa$, and in matching sessions we copy K_2 to K_1, as Game 5 copy k_1 to k_2 or vice versa in such sessions. Since there're at most q_{ses} such sessions, and from security of kdf, the difference between Game 5 and Game 6 is negligble to $\mathcal{Z}$, i.e. $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}} \epsilon_{\text{kdf}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$

Game 7 (Ideal-world game): This is the ideal-world interaction, i.e. an interaction of environment $\mathcal{Z}$ with simulator SIM and functionality $\mathcal{F}_{\mathsf{aPAKE}}$, shown in Figure 6.11.

Observe that Game 6 is identical to the ideal-world Game 8. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof. $\square$

5.6 Lattice-Based UC PAKE from EKE and Saber KEM

We argue that the CPA-secure Key Encapsulation Mechanism (KEM) at the heart of the Saber [56] public key encryption, whose security is based on the Module-LWR problem, achieves also the *anonymity* and *uniform public keys*property, see Section 2, under the same Module-LWR assumption. In Figure 5.22 we show the EKE-KEM construction, which is Figure 5.15 instantiated with Saber KEM. Note that Theorem 5.3 implies that the resulting protocol is a UC PAKE under the Module-LWR assumption.

Saber Cryptosystem. We define the notation needed to introduce Saber. Let $\mathbb{Z}_q$ be the ring of integers modulo q represented in $[-q/2+1, q/2]$ and R_q a polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$, where n is a power of 2 and a security parameter (and a length of the session key output by Saber). Let $R_q^{l_1 \times l_2}$ be the ring of l_1 by l_2 matrices over R_q. (Below we use uppercase bold font to denote matrices and lowercase bold font to denote vectors.) Let $\mathcal{U}(R_q)$ be a uniform distribution over R_q and let $\chi_\mu(R_q)$ be a distribution where each polynomial coefficient is chosen from a binomial distribution centered at 0 with parameter μ (and standard deviation $\sqrt{\mu/2}$). When these distributions are taken over a matrix space $R_q^{l_1 \times l_2}$ instead of R_q, this stands for choosing each matrix entry (or vector if $l_2 = 1$) according to that distribution.

Denote $\lfloor \cdot \rfloor$ as flooring to the nearest lower integer and $\lfloor \cdot \rceil$ as rounding to the nearest integer. The operation $\lfloor \cdot \rfloor_{q \to p}$ takes an integer $x \in \mathbb{Z}_q$ as input and outputs $\lfloor p/q \cdot x \rfloor \in \mathbb{Z}_p$, and similarly $\lfloor x \rceil_{q \to p} = \lfloor p/q \cdot x \rceil \in \mathbb{Z}_p$. We use $[\cdot]_p$ to denote mod p operation. Saber uses moduli $q = 2^{\epsilon_q}, p = 2^{\epsilon_p}, T = 2^{\epsilon_T}$ with $q > p > T$, and the constants added in $\lfloor \cdot \rfloor$ in Figure 5.22 are set as $h_1 = \frac{q}{2p} \in R_p, h_2 = \frac{p}{4} - \frac{p}{2T} + \frac{q}{2p} \in R_p$ and $\mathbf{h} = \frac{q}{2p} \in R_q^{l \times 1}$. (Saber NIST proposal [56] suggests parameters $\epsilon_q = 13, \epsilon_p = 10, \epsilon_T = 4$.)

Security of Saber relies on the hardness of the Module Learning with Rounding problem (Mod-LWR)[20], defined as a variant of the Learning with Errors (LWE) problem where

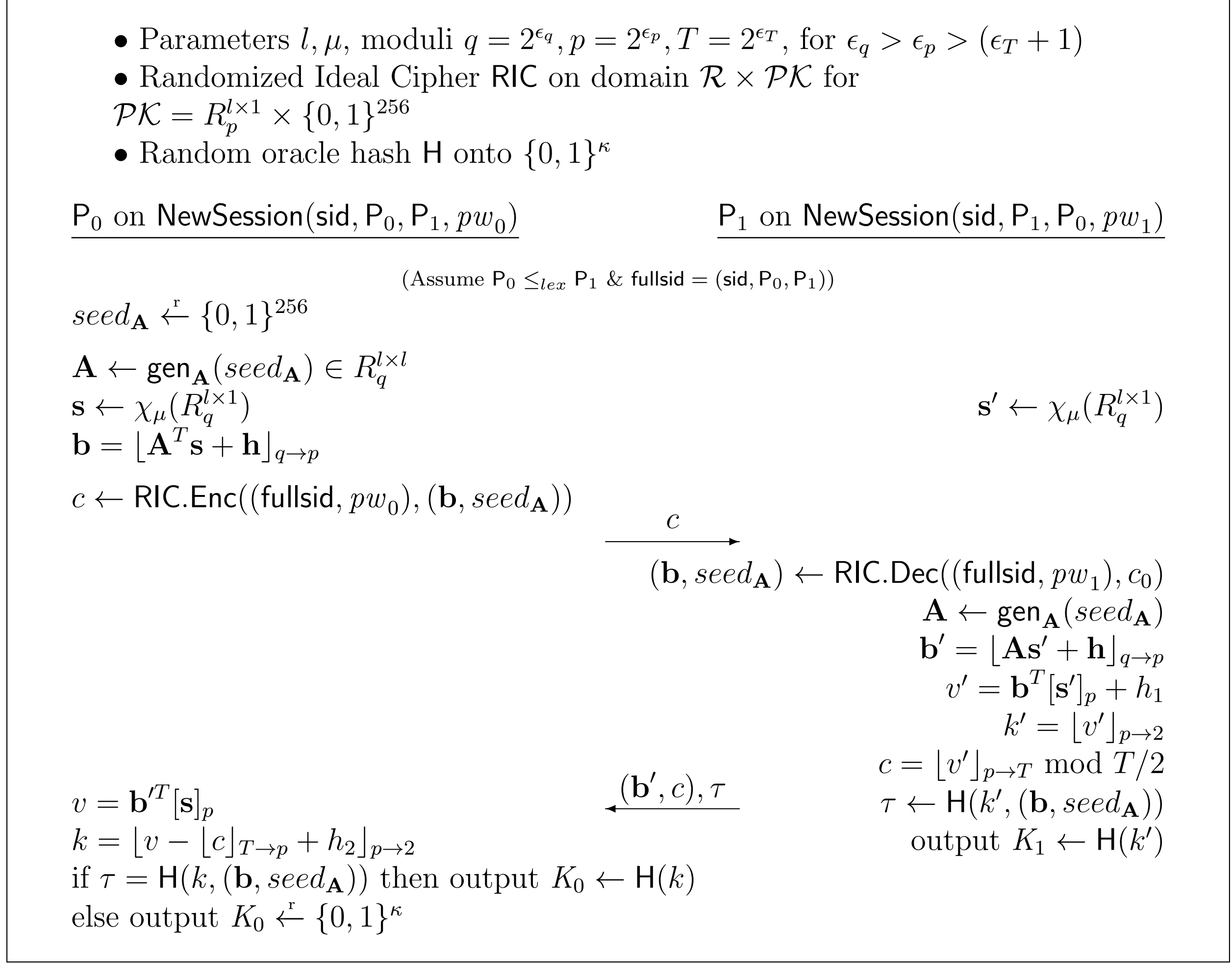


Figure 5.22: Protocol EKE-KEM of Section 5.4.1 instantiated with Saber KEM

the error is implicitly generated by the integer rounding operation. The advantage of a polynimal-time adversary $\mathcal{A}$ against the generalized Mod-LWR problem is defined as follows for parameters m, l, p, q, μ s.t. $p < q$:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{Mod-LWR}}_{m,l,q,p,\mu}(\mathcal{A}) \;=\; \big|\; &\Pr[\, 1 \leftarrow \mathcal{A}(\mathbf{A}, \lfloor \mathbf{As} \rceil_{q\to p}) \;:\; \mathbf{s} \xleftarrow{r} \chi_\mu(R_q^{l\times 1}); \mathbf{A} \xleftarrow{r} \mathcal{U}(R_q^{m\times l}) \,] \\ &- \Pr[\, 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{u}) \;:\; \mathbf{A} \xleftarrow{r} \mathcal{U}(R_q^{m\times l}); \mathbf{u} \leftarrow \mathcal{U}(R_p^{m\times 1}) \,] \;\big|\end{aligned}$$

EKE instantiated with Saber. In Figure 5.22 we show the EKE-KEM protocol of Sec-

tion 5.4.1 instantiated with Saber KEM. The resulting protocol is essentially a Saber key exchange protocol but with the initiator's public key encrypted using Randomized Ideal Cipher , and with the responder attaching a key-and-password confirmation message.

The following theorem, proven in [56], states the CPA security of Saber under the Mod-LWR assumption:

Theorem 5.5. *Assuming* $\mathsf{gen}_{\boldsymbol{A}}$ *to be a random oracle. For any adversary* $\mathcal{A}$*, there exists two adversaries* B_1 *and* B_2*, such that:*

$$\boldsymbol{Adv}_{Saber}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) \leq \boldsymbol{Adv}_{l,l,\mu,q,p}^{\mathsf{mod-lwr}}(B_1) + \boldsymbol{Adv}_{l+1,l,\mu,q,p}^{\mathsf{mod-lwr}}(B_2) \quad \textit{if} \;\; q/p \leq p/T.$$

The two further KEM properties needed in the $\mathsf{EKE\text{-}KEM}$ protocol of Section 5.4.1 are ciphertext anonymity and uniform public keys (see Section 2.2 for definition of these notions), but Saber satisfies these properties under the same Mod-LWR assumption:

Theorem 5.6. *Saber KEM satisfies the* uniform public keys *property on domain* $\mathcal{PK}$ *and the* anonymity *property under Module-LWR assumption.*

Proof. Below we sketch the proof of Theorem 5.6. The uniform public keys property which requires the public key generated to be indistinguishable from uniform, is by definition of Module-LWR problem and proved in Game 2 in the same proof of Theorem 5.5, where $\mathbf{b}$ is replaced with a uniform value. The anonymity property, which requires that given two different public keys and a ciphertext $(\mathbf{b}', c)$ generated by one of them, it's computationally hard to distinguish the correct key, is also satisfied by Saber since without information about secret $\mathbf{s}'$, LWR samples ($\mathbf{A}, \mathbf{b}$') and $(\mathbf{b}, v')$ are both indistinguishable from random elements by definition of LWR. The full proof is given in [106]. □

Comparison with prior lattice-based PAKEs. We recall prior work on lattice-based PAKE's to compare it to the EKE-KEM(Saber) protocol shown in Figure 5.22. The short summary is that EKE-KEM(Saber) appears to be the first UC PAKE from lattice assumption, and it also forms a two-round PAKE which has the smallest bandwidth among prior lattice-based PAKE proposals. Indeed, its bandwidth is minimal because it adds only 3κ bits to the underlying (plain) Key Exchange implemented by KEM.

The first lattice-based PAKE was shown by Katz and Vaikuntanathan [94], where both parties send a CCA-encrypted ciphertext to each other, compute Approximate Smooth Projective Hash (ASPH) values on ciphertexts, and conduct a key reconciliation subprotocol to derive a session key. This protocol needs three rounds and the underlying CCA-encrypted ciphertext actually contains n CPA-encrypted ciphertexts, which is costly to compute. KV is further optimized by Zhang and Yu [125], who proposed a 2-round PAKE with a new ASPH based on a "splittable CCA-secure encryption". Following the same track, Benhamouda [30] adapts Groce and Katz [73] framework using KV's realization of ASPH and as result, gets new 3-round and 2-round PAKEs in standard model, and they further optimize the protocol to one round, using the same SS-NIZK approach as in [125]. However, construction of lattice-based SS-NIZK in standard model appears to be still an open question. Moreover, all of these works rely on standard-model CCA-secure encryption which appears expensive to realize. We refer for more details to [91], who explain the effiency challenges in this line of work.

[91] is the first to construct a lattice-based PAKE in the standard model which only requires CPA-secure encryption, and it's significantly more efficient compared compared to the PAKEs which use CCA-secure lattice-based encryption. Ding et al. [59] proposed a still much more efficient scheme assuming ROM. Their scheme appears to be a lattice-based counterpart to the PPK protocol of Boyko et al. [42], and thus also to EKE. The significant difference, however, is that in PPK hashed password is used as a one-time mask on the KE

messages, wheres in EKE it is used as key that encrypts the KE messages using an ideal cipher. Consequently, Ding et al. [59] analyze the security of their PAKE in the "BMP" model of [42], whereas we analyze our proposal in the UC PAKE model. (We note, however, that the BMP model for PAKE is mostly likely equivalent to the recently proposed UC *relaxed PAKE* model [5].) Apart of this difference in analysis, the fact that our analysis uses KEM as a black-box allows instant reuse of efficiency improvements in lattice-based KEMs. Indeed, Saber uses a much smaller field modulus $q = 2^{13}$ compared to $2^{32} - 1$ in [59], which reduces the size of both the KEM public key and the ciphertext (and these sizes are further reduced by rounding operations).[17]. We benchmark the bandwidth for the last three lattice PAKEs discussed above, which seem to form the most efficient proposals. For security parameter $\kappa = 128$, the total bandwidth is 207 KB for [91], 8.32 KB for [59] and 1.376 KB for EKE-KEM(Saber).

Table 5.6 provides a detailed comparison on efficiency of these last three lattice PAKEs.

Scheme	Bndw (KB)	Rounds	Assum	Security	Model
JGHNW[91]	207	3	(R)LWE	BPR	Standard
Ding17[59]	8.320	2	(R)LWE PairWE	Bokyo[42]	ROM
EKE-KEM (Saber)	1.376	2	LWR	UC	ROM

Table 5.1: Comparison of lattice-based PAKE protocols based on bandwidth, rounds, security assumptions, security claims, and security model

[17]Saber[56] authors argue that this more aggressive parameter suffices in their construction, and while using large prime moduli can possibly adopt Number Theoretic Transformation (NTT) to speedup polynomial multiplications, [56] using power-of-two moduli has its own advantages including: (1) avoiding modular reduction and rejection sampling; (2) the use of LWR halves the amount of randomness required compared to LWE-based schemes, and thus reduces bandwidth; (3) the module structure provides flexibility by reusing one core component for multiple security levels. See more details in [56]

Chapter 6

Generic compiler from PAKE to asymmetric PAKE using KEM

6.1 Introduction

Password Authenticated Key Exchange (PAKE) [28] allows two parties to establish a high-entropy session key for secure communication, if and only if they share the same password. The *asymmetric (or augmented) PAKE* (aPAKE) [29] used for client-server communication, is a variant of PAKE where the server stores a one-way function of the password, and a shared session key is established if and only if the client enters the correct pre-image of the server's input. Currently used PKI-based "password-over-TLS" authentication has multiple vulnerabilities which can lead to password disclosures: An active attacker learns the password if the client falls victim to the so-called *phishing* attack, and the password appears in the cleartext on the server side during protocol execution. This motivates the development of aPAKE protocols and their integration with TLS1.3 in the ongoing standardization by the Internet Engineering Task Force [120].

In the classic computational model there are many works both on PAKE, e.g. [11, 114, 76, 23] in the Random Oracle Model (ROM) and [93, 71] in the standard model, and aPAKE, e.g. [72, 88, 74, 68]. Some PAKE schemes were also shown secure specifically under post-quantum assumptions. These include PAKE proposals using code-based cryptography [36] and isogeny-based group actions [9], but PAKE constructions based on lattice assumptions have received more attention and are closer to practicality. These include constructions in the standard model, e.g. [94, 125, 30, 91, 12], and constructions that assume the random oracle model [59, 67, 22, 19], which are much more efficient. In particular, [67, 22, 19] are generic constructions of *universally composable* (UC) PAKE from any KEM which is (weak) anonymous and PCA secure (i.e. Plaintext-Checking-Attack secure). These requirements are satisfied by Kyber [40], a post-quantum KEM selected by NIST [3], secure under LWE assumption in ROM. The cost of the resulting UC PAKE is dominated by one cycle of KEM key generation, encryption, and decryption, and the bandwidth can be approximated as a single pair of KEM public key and a ciphertext.

However, regarding UC aPAKE's secure under PQ assumptions, to the best of our knowledge the only known methods are implied by generic aPAKE constructions, if all the building blocks required by a given construction are instantiated with post-quantum secure cryptosystems. The first group of such constructions are generic PAKE-to-aPAKE compilers [72, 84] which can be instantiated using post-quantum building blocks, but the resulting cost of such aPAKE would be at least 4x larger than LWE-based PAKEs mentioned above. Other aPAKE constructions include the OPAQUE protocol [88], which builds a (strong) aPAKE from OPRF and AKE, and the KHAPE and OKAPE protocols [74, 68], which build aPAKE from key-hiding AKE. However, post-quantum realizations of these building blocks either do not yet exist or are much less efficient than LWE-based KEM like Kyber. We discuss these points in more detail below.

6.1.1 Prior aPAKE Constructions

There are three prior generic constructions which convert any UC PAKE to UC aPAKE using some additional cryptographic tool. Gentry, MacKenzie and Ramzan [72] showed such compiler using signatures, and Hwang et al. [84] showed two such compilers, one using non-interactive zero-knowledge proof of knowledge (NIZK), the other using KEM. We discuss their implications to aPAKE based on PQ assumptions below.

PAKE-to-aPAKE compilers using Signatures or NIZK's. The PAKE-to-aPAKE compiler of [72], called the Ω-method, shown in Figure 6.1, transforms any UC PAKE into UC aPAKE in ROM using a secure signature scheme. Note that parties C and S in Figure 6.1 interact with a box denoted PAKE: This stands for running any UC-secure PAKE in a black-box way. Each party forms inputs to this UC PAKE subprotocol using a unique *session identifier* sid, the identifiers of their own party and their counterparty, e.g. C and S for the client, and string h used as a PAKE password. In addition to identifiers $\mathsf{sid}, \mathsf{C}, \mathsf{S}$, the top-layer aPAKE protocol inputs include password pw for client C, and a *user account identifier* uid for both parties, which server S uses to retrieve previously created *password file* $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$. Intuitively, aPAKE can be secure only if $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ stores a one-way function of the password, which leaks the password only via a successful brute-force *offline dictionary attack*. This is true in the Ω-method, where $\mathsf{file}[\mathsf{uid}, \mathsf{S}] = (h, pk, \bar{e})$, because h and $\bar{e}$ are derived via RO hash functions involving password pw, and they leak no information unless the adversary queries these hash functions on inputs that include pw. Importantly, the Ω-method relies on sub-protocol PAKE which in addition to session key K also outputs its transcript tr. The only assumption on these transcripts is that they are non-colliding.

The main idea of the Ω-method is that C signs PAKE transcript tr using a signature key retrieved from S via a decryption that involves both password pw and key k_e derived from PAKE output K. For this purpose the protocol uses two specific symmetric encryption

- signature scheme (Sig.KG, Sign, Verify), with secret keys of size $s[\kappa]$
- symmetric encryption scheme (OTP.E, OTP.D) s.t. $\mathsf{OTP.E}_k(m) = \mathsf{H}(k) \oplus m$
- authenticated encryption (AEnc, ADec) s.t. $\mathsf{AEnc}_k(m) = \mathsf{H}(k) \oplus (m|\mathsf{H}'(m))$
- hash functions $\mathsf{H}, \mathsf{H}', \mathsf{H}''$ with ranges resp. $\{0,1\}^{\kappa+s[\kappa]}$, $\{0,1\}^{\kappa}$, $\{0,1\}^{\kappa}$
- pseudorandom function prf with range $\{0,1\}^{\kappa}$

Password File Initialization on S's input $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$:

S sets $h \leftarrow \mathsf{H}''(\mathsf{S}, \mathsf{uid}, pw)$, generates keys $(sk, pk) \leftarrow \mathsf{Sig.KG}(1^{\kappa})$ and ciphertext $\bar{e} \leftarrow \mathsf{AEnc}_{pw}(sk)$, stores $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h, pk, \bar{e})$, and discards all other values.

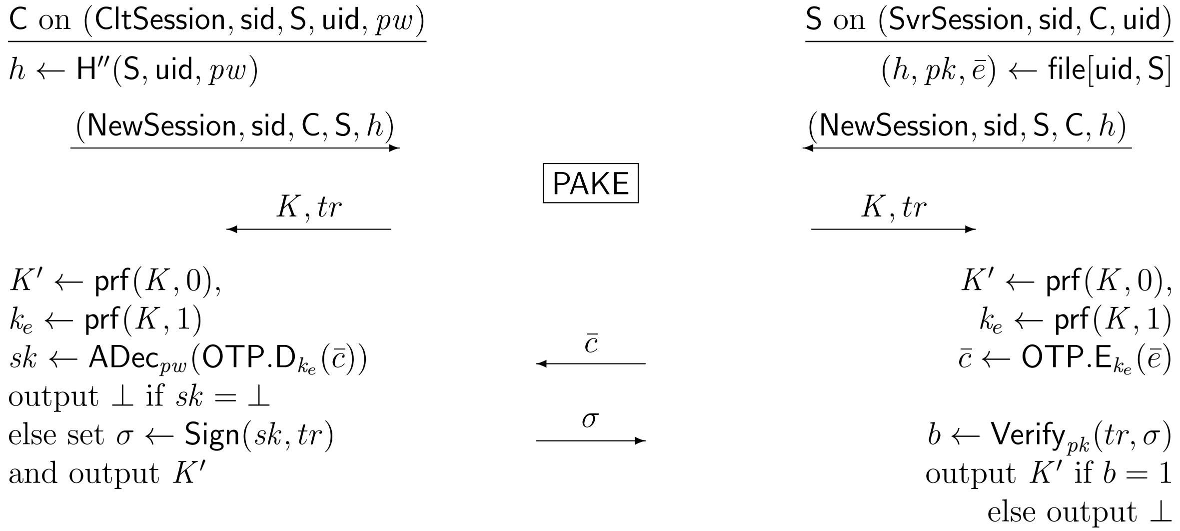


Figure 6.1: Ω-method: PAKE to aPAKE compiler using Signatures [72]

schemes OTP.E and AEnc, where OTP.E is simply a one-time pad and AEnc is a one-time pad of plaintext concatenated with its hash. Intuitively, Ω-method works because if an adversary doesn't enter the correct password hash h into a PAKE instance then it has no information about k_e, hence it learn nothing from S's ciphertext $\bar{c}$ (which is encrypted under k_e), and cannot send any ciphertext $\bar{c}$ of its own to C s.t. $\mathsf{AD}_{pw}(D_{k_e}(\bar{c})) \neq \perp$, because that happens only if $\bar{c} = \mathsf{H}(pw) \oplus \mathsf{H}(k_e) \oplus (m|\mathsf{H}'(m))$ for some m. Moreover, if adversary $\mathcal{A}$ corrupts S and learns $\mathsf{file}[\mathsf{uid}, \mathsf{S}] = (h, pk, \bar{e})$, it can use it to authenticate to C, but without hashing the right pw it cannot learn $sk = \mathsf{AD}_{pw}(\bar{e})$. Furthermore, CMA-security of signature implies that $\mathcal{A}$'s interaction with any number of C sessions, which leak $\mathsf{Sign}(sk, tr)$ for any transcript tr of a PAKE session established between $\mathcal{A}$ and C, does not let the adversary

forge a signature σ' on transcript tr' of a PAKE session established between $\mathcal{A}$ and S.

The Ω-method adds 2 flows to the underlying PAKE. However, assuming PAKE constructed from KEM following [67, 22, 19], the PAKE sub-protocol takes only 2 flows, hence the 1st flow of the Ω-method can be piggybacked on the 2nd flow of the PAKE, in which case the resulting UC aPAKE takes only 3 communication rounds (and provides implicit authentication). The computation overhead is dominated by signature generation for C and verification for S, and the bandwidth overhead by the sizes of the signing key and the signature.[1]

A variant of the Ω-method was shown by Hwang et al. [84], with signature replaced by a ROM-based NIZK of $sk = \mathsf{H}(pw)$ where $pk = \mathsf{OWF}(sk)$ is held by the server. This construction adds only one C-to-S flow to PAKE, with the PAKE key used to encrypt the NIZK, and the PAKE transcript used as a NIZK label to enforce a binding between the NIZK and a PAKE session. However, if PAKE is instantiated as above then this aPAKE would still require 3 communication rounds, if C is the initiator. Moreover, current NIZK's of preimage of a PQ one-way function are significantly more expensive than PQ signatures, see e.g. [104] for a NIZK PoK of preimage of the LWE one-way function.

Performance Considerations. NIST [3] selected Dilithium signature [62] as a primary post-quantum signature method, with Falcon [116] and Sphincs+ [83] as secondary options. Performence benchmarks comparing Kyber KEM and Dilithium signature are reported in [102] and comparisons to Falcon signatures are included in [103]. For 196-bit security, the size of Dilithium signature is 2700 bytes and the size of the secret key is roughly twice that, while the sizes of public keys and ciphertexts in the CCA-secure KEM of Kyber are resp. 1088 and 1184 bytes. The running time of Kyber KEM (kg, enc, dec) in CPU cycles are respectively (85k, 125k, 135k), while Dilithium (kg, sign, ver) are (250k, 1000k, 300k). Compared to Dilithium, Falcon has 4x smaller signature size but its signing operation is at

[1] The encrypted signing key sk can be omitted if C regenerates sk using password hash as key generation randomness. This introduces performance overheads which vary depending on the specifics of the key generation algorithm.

least 3.5x slower and verification 5.4x faster [103].[2]

It follows that the costs of KEM-based UC PAKE [67, 22, 19] instantiated with Kyber KEM can be estimated at 2272 bytes of bandwidth and 345k CPU cycles, counting the costs of only Kyber KEM component of such PAKE, while the overhead of the Ω-method PAKE-to-aPAKE compiler instantiated with Dilithium signature would be at least 2700 bytes of bandwidth (counting only signature size, although this would increase computation costs, see footnote 1) and 1300k CPU cycles, resulting in UC aPAKE which is a factor of 2.2x ($1 + 2700/2272$) and 4.8x ($1 + 1300/345$) more expensive than the underlying UC PAKE in respectively bandwidth and computation. If Ω-method is instantiated with Falcon signatures then the factors of bandwidth and computation increase between the resulting aPAKE and the underlying PAKE would be respectively at least 1.3x and 14.4x. (We omit Sphincs+ because it has much larger signature sizes still, together with larger computation costs.)

PAKE-to-aPAKE compiler based on Diffie-Hellman KEM. Hwang et al. [84] showed another PAKE-to-aPAKE compiler, denoted $\mathsf{HJK}^+(2)$ and shown in Figure 6.2, which uses KEM instead of signature or NIZK to implement client-to-server authentication necessary in aPAKE. In [84] this construction was using specifically the DH-based KEM, but we use abstract KEM notation to highlight KEM properties needed for this construction. Following [84], we assume that the KEM secret key sk is chosen at random from some domain and that pk is derived from sk via a deterministic algorithm $\mathcal{PK}$. (Without loss of generality, sk can be the randomness of the key generation algorithm, and as mentioned in footnote 1 the same can be done for signatures in the Ω-method.)

Compiler $\mathsf{HJK}^+(2)$ is simple: The public key pk in the server's password file is a KEM public key, and after generating session key K and transcript tr by a PAKE instance on a password hash h (this part is the same as in the Ω-method), the server sends a challenge

[2]The 3.5x factor in signing cost increase is a lower-bound because [103] compares performance of Falcon with 128-bit security to Dilithium with 196-bit security.

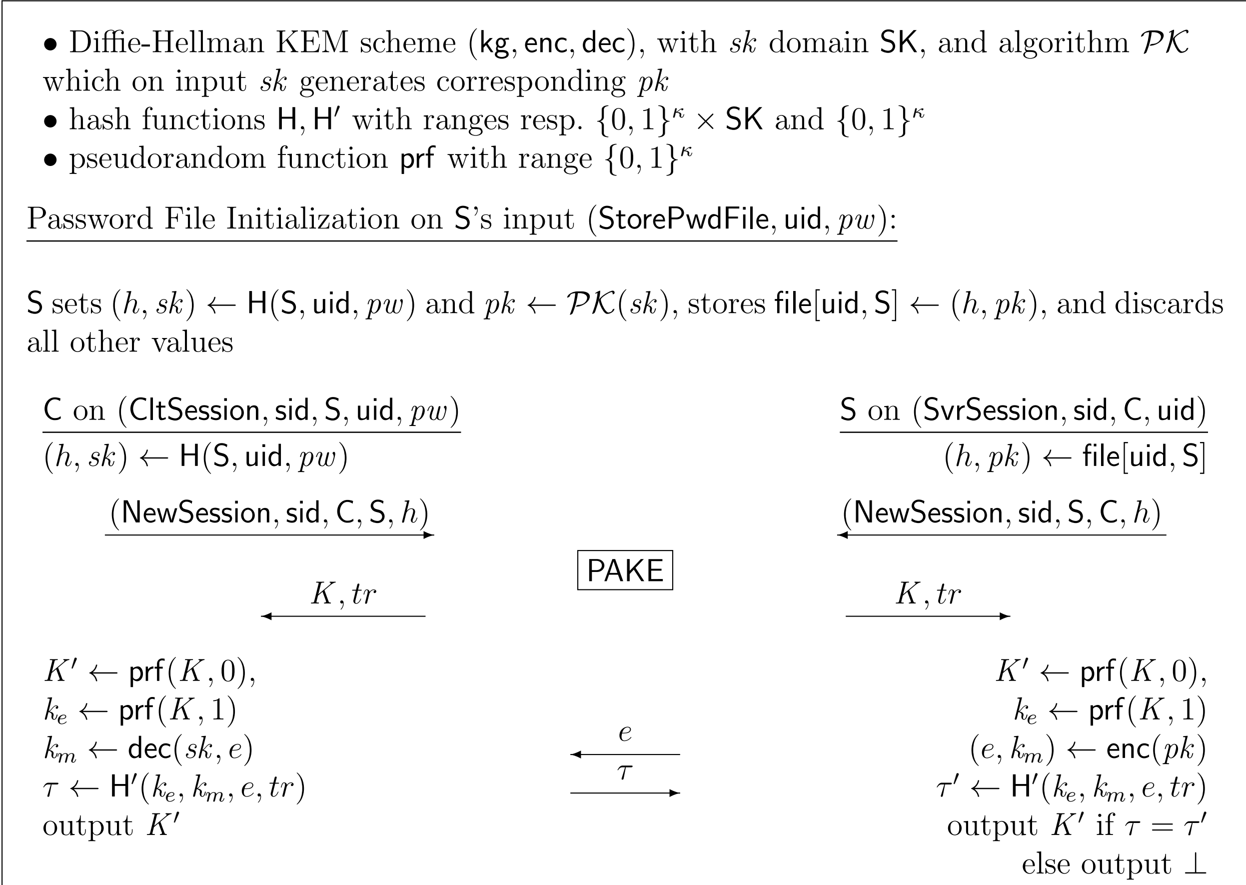


Figure 6.2: Protocol $\mathsf{HJK}^{+}(2)$: PAKE to aPAKE complier using DH KEM [84]

KEM ciphertext e generated from pk to the client, and accepts if the client sends back a key confirmation message τ computed as an RO hash of the protocol transcript (tr, e), key k_e derived from the PAKE output K, and key k_m decrypted from ciphertext e.

The authors of [84] did not claim security for general KEMs but it appears that this compiler is secure if instantiated with any KEM that satisfies *strong anonymity* and *one-wayness under Plaintext Checking Attack* (OW-PCA). Strong anonymity requires that KEM ciphertext e cannot be linked to a KEM public key used in creating it even if the attacker knows the corresponding *secret key*. (See Section 2 for formal definitions.) Strong anonymity is needed because the KEM secret key is derived from a password hash, hence the adversary can compute it off-line and test it against the KEM ciphertext e sent by the server. OW-PCA security of KEM is required for similar reasons as CMA security of signature in the Ω-method: If an adversary compromises the server and learns hash h and public key pk, then

it can learn key K from a PAKE subprotocol executed on h, and it can then use the client session holding key sk as a Plaintext-Checking Oracle: To test if a ciphertext,plaintext pair (e', k'_m) is correct, the adversary sends e' to the client and conclude that it is correct if the authenticator τ sent by the client is equal to $\mathsf{H}'(k_e, k'_m, e', tr)$.

Both of these requirements can be realized by the *hashed* Diffie-Hellman KEM: Strong anonymity holds because DH KEM ciphertext is a random group element independent of the KEM key, while OW-PCA property of *hashed* DH KEM is equivalent to the GapDH assumption, i.e. that solving the Computational DH problem is hard even given access to a Decisional DH oracle.[3] However, it is not clear how to achieve both properties with e.g. LWE-based KEM, and to the best of our knowledge none of the NIST KEM candidates satisfy them. Hashed Diffie-Hellman KEM achieves OW-PCA under GapDH because it is actually IND-CCA secure under the same assumption, which is a basis of the DHIES IND-CCA public key encryption [6]. However, we do not know if LWE-based cryptosystems can be secure under a corresponding assumption, and to the best of our knowledge there is no DHIES equivalent based on LWE. Indeed, all post-quantum NIST KEM proposals achieve CCA-security by applying the Fujisaki-Okamoto transform [70] to the underlying CPA-secure KEM. In this method the underlying KEM is used in a circular way, utilizing ROM: The ciphertext effectively encrypts the random seed which was used to generate it, and the decryptor accepts a ciphertext as valid only if it can regenerate it using the decrypted randomness. Unfortunately, this method also directly contradicts the strong anonymity requirement because this validity test holds only using the correct secret key.

Other aPAKE constructions. Another aPAKE construction type is OPAQUE [88], which achieves a *strong* aPAKE i.e. aPAKE which in addition is secure against offline dictionary attacks made before server corruption. The key ingredient of the OPAQUE protocol is an Oblivious Pseudorandom Function subprotocol (OPRF), and there is much active work on

[3] In particular, the security claim in [84] regarding $\mathsf{HJK}^+(2)$ appears incorrect: They claim security under CDH assumption, but it appears that GapDH is necessary.

OPRFs secure under lattice or isogeny assumptions [15, 14, 21, 37], but so far the constructions are far less efficient than KEMs constructed under the same assumptions.[4] Another type of aPAKE constructions are KHAPE [74] and OKAPE [68] schemes, which are generic constructions from any *key-hiding* AKE. Such key-hiding AKE can be instantiated with SKEME, which is built solely from KEM. However, similarly to the case of aPAKE construction $\mathsf{HJK}^{+}(2)$, SKEME satisfies the key-hiding AKE property needed in [74, 68] only if it is built from KEM that satisfies both strong anonymity and OW-PCA security. Consequently, these constructions are secure under GapDH if instantiated with (hashed) Diffie-Hellman KEM, but it is an open problem to construct such KEM (and a key-hiding AKE) based on post-quantum assumptions. [119] provides another type of aPAKE construction based on CDH assumption, and it also remains open how to instantiate it with e.g. LWE.

6.1.2 Our Contributions

In this work we present a new aPAKE compiler as shown in Figure 6.3, which converts any UC-secure symmetric PAKE into a UC-secure asymmetric PAKE. This compiler is a general one which can be initiated efficiently upon quantum-hard problems. We now discuss our results and compare the efficiency with previous compilers in detail.

Compared to previous aPAKE compilers, the significant difference is that our compiler relies on a different construction paradigm, namely PAKE+KEM. Since we leverage the performance advantage of lattice KEMs, our compiler overcomes the current computation inefficiency of lattice signatures or lattice NIZK, which are the core building blocks of previous PAKE-to-aPAKE compilers. Our result shows that as long as the underlying KEM is CCA-secure (which is satisfied by all lattice KEMs currently being considered for standardization), and the given PAKE protocol is a UC realization of PAKE functionality [72] as shown in

[4] Efficient post-quantum aPAKE would be attractive even if post-quantum OPRF were practical, because OPRF used together aPAKE as shown in [88], realizes a strong aPAKE with an attractive property, not present in OPAQUE, that the server is the first to learn if an authentication attempt succeeds.

Figure 2.1, then through our compiler the resultant aPAKE protocol realizes the UC aPAKE functionality in Figure 2.4.

Our compiler derives the secret key of KEM scheme from a random oracle hash onto the client's password, which saves storage space on the server side. This can be realized in Diffi-Hellman KEM and most lattice based KEMs (e.g. Kyber, since in these KEM schemes the secret keys are in $\mathbb{R}_p^k$, i.e. an array of ring elements which you can derive from hash) and isogeny-based KEMs (where secret keys are randomly sampled from a group). In addition we also present a key-generation oblivious variant of our general PAKE-to-aPAKE compiler in Figure 6.8, where similar to GMR compiler, the client doesn't directly derives KEM's secret key from hash on password, instead server stores the encrypted secret key under password, and send to client during online sessions who later decrypts with the correct password. While this variant reduces the computation cost on server side during online phase at the cost of extra storage, the main reason for this variant is to further increase the generality of the compiler so it can also fit cryptographic schemes whose secret keys are hard to derive from hash, such as RSA signatures/KEMs.

Note that although our compiler adds two additional messages to the underlying PAKE, they can be reduced to only one round of communication if the last message in symmetric PAKE goes from server to client. To see this more clearly, we show an efficient instantiation of our compiler in Figure 6.9, where the underlying symmetric PAKE satisfies this "last message goes S-to-C" property, and the resultant aPAKE protocol only requires three message flows.

While the KEM part is the core building block of our compiler and can be instantiated with DH-KEM using classical assumption, or any CCA-secure lattice KEM for post-quantum security, our compiler does require other building blocks, including message authentication code (MAC) and authenticated encryption (AE). However, these other building blocks can be easily and efficiently realized with existing tools, even for quantum resistance purpose. For example, MAC can be instantiated using HMAC with a sufficiently long key (e.g. 128

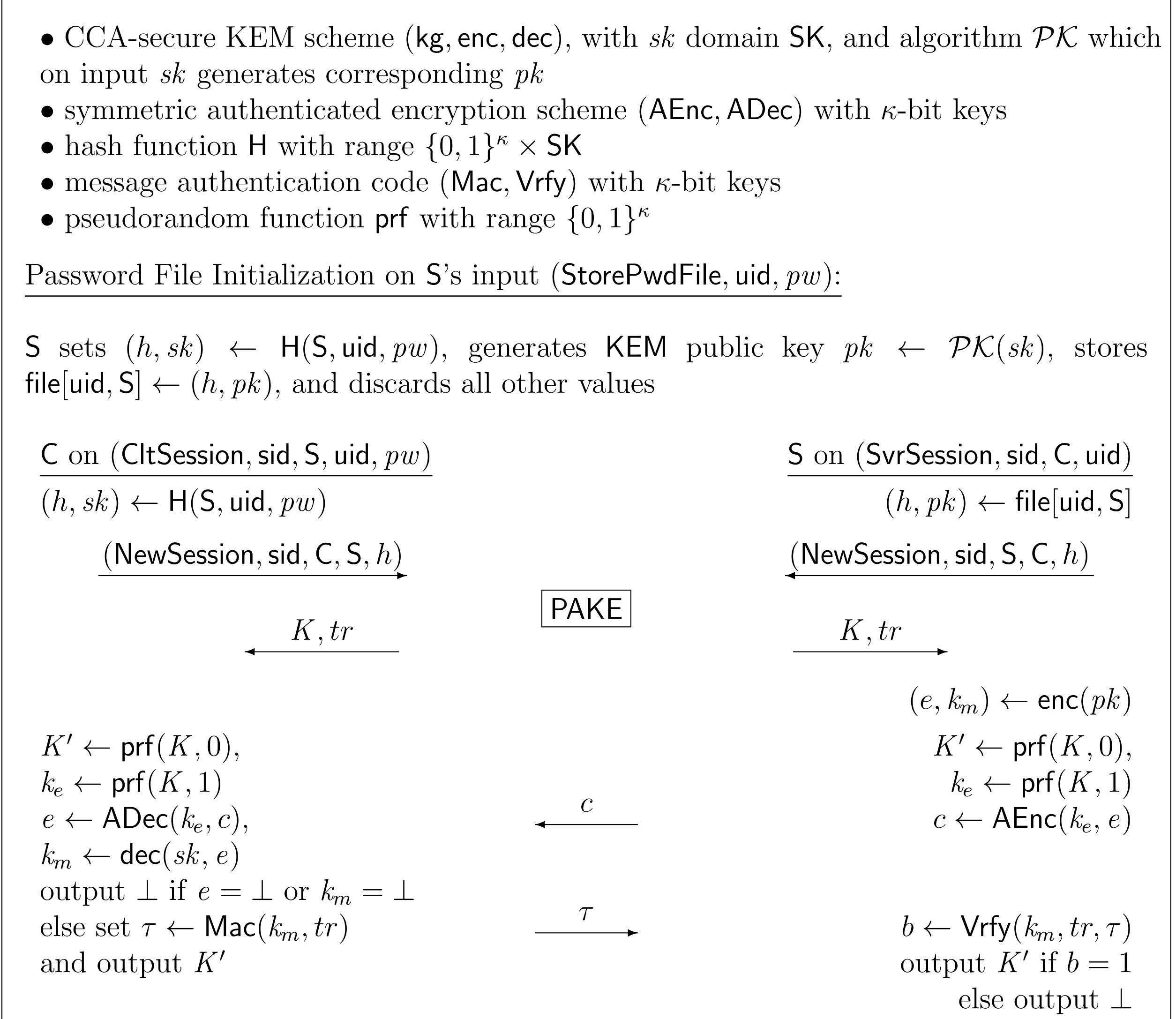


Figure 6.3: Protocol APAKEM: PAKE to aPAKE compiler using CCA-secure KEM

or 160 bit). AE, with random-key robustness and authenticity properties, can be achieved using encrypt-then-MAC paradigm, where the MAC part needs to be collision resistant with respect to the message and the key, which can also be instantiated with HMAC.

In addition, the concrete instantiation of our compiler shown in Figure 6.9, gives the first aPAKE that can be solely built upon KEM, i.e. a KEM-to-aPAKE compiler. That said, for post-quantum purpose, only the KEM part needs to be based on quantum-resistant assumptions, which can efficiently realized by CCA-secure lattice KEM, e.g. Kyber, or CCA-secure isogeny KEM, e.g. group action based KEM from Hashed ElGamma method. This feature

also offers significant advantages for practical deployment, since its modularity and generality allows its use with different KEM schemes from different assumptions, which provide different features and performance tradeoffs. In our concrete instantiation, the underlying symmetric PAKE also relies on KEM, but only requires a weak anonymity 2.6 property, which is defined by [67] and proven to be realized directly by the key exchange protocol of Kyber and Saber, and many other existing lattice KEM schemes which only requires CPA security. This offers additional modularity in protocol design level, which allows both "strong" and "weak" secure KEMs to be implemented using the code library from the same scheme, which is more friendly for developers.

Scheme	Reference	Round complexity[(1)]	Lattice-based instantiation	Building blocks	Security model
Ω-method	[72]	3	✓	PAKE+Sign	ROM
HJK+(1)	[84]	3	✓[(2)]	PAKE+NIZK	ROM
HJK+(2)	[84]	3	✗	PAKE+DDH	ROM
OPAQUE	[88]	3	✓[(3)]	OPRF+AKE	ROM
KHAPE	[74]	3	✗	kh-AKE[(4)]	ROM, IC
OKAPE	[68]	2	✗	kh-AKE[(4)]	ROM, IC
APAKEM	this paper	3	✓	PAKE+KEM	ROM

Table 6.1: Comparison of UC aPAKE constructions. *Comments:* [(1)]For all PAKE-to-aPAKE results we assume two-round PAKE instantiated from LWE [67, 22, 19]; [(2)]Given current LWE-based NIZK's this scheme is not more efficient than Ω-method; [(3)]Current lattice-based OPRF's are significantly more costly than KEM's; [(4)]kh-AKE stands for *key-hiding AKE*, for which there are no current lattice-based solutions;

Note on Quantum Attackers and QROM. In this work we only analyze security of the proposed scheme against a classic computation adversary as opposed to a quantum computation adversary. In particular, we do not consider adversarial access to quantum random oracle because. Note that neither the KEM-to-PAKE compilers [67, 22, 19] nor the prior PAKE-to-aPAKE compilers [72, 84] analyze their security in the quantum setting.

Due to technical difficulties, there has been very limited work that considers quantum adversaries in UC framework, and even less for QROM, and currently the best we can do is to

prove the security assuming ROM. We also don't observe any other work on aPAKE that is proven secure assuming QROM, either in UC model or in BPR model (where we expect QROM to be easier to deal with), and we leave that as future work.

Performance Considerations. As discussed earlier, the costs of KEM-based UC PAKE [67, 22, 19] instantiated with Kyber KEM can be estimated at 2272 bytes of bandwidth and 345k CPU cycles. The overhead of moving from PAKE to aPAKE created by our compiler instantiated with the same Kyber KEM is around 1184 bytes of bandwidth and 260k CPU cycles (again counting only the KEM overheads), resulting in UC aPAKE which is a factor of 1.5x and 1.8x more expensive than the underlying UC PAKE in resp. bandwidth and computation. Recall from the performance discussion few pages back that the corresponding costs of aPAKE constructed via the Ω-method, counted as fractional increase compared to the UC PAKE, were 2.2x and 4.8x if Ω-method uses Dillithium signatures, and at least 1.3x and 14.4x if it uses Falcon signatures. This implies that our KEM-based aPAKE will use roughly 1.5x (2.2/1.5) less bandwidth and 2.7x (4.8/1.8) less computation than the aPAKE of [72] instantiated with Dilithium signatures, and roughly the same bandwidth but at least 8x (14.4/1.8) less computation than [72] instantiated with Falcon signatures.

Note that although our analysis focus on the performance advantage of our compiler over other aPAKE compilers built on lattice assumptions, our compiler is a general PAKE-to-aPAKE compiler, and even if built on classical assumptions or other quantum-resistant assumptions such as isogeny, our compiler is still very efficient compared to other existing methods. Table 6.1 provides a detailed comparison on efficiency of the existing aPAKE compilers versus ours[5][6].

[5] Note that we do not compare computation cost here because it depends on the cryptographic assumption and the concrete instantiation for the underlying building blocks, however we give a detailed discussion on computation cost of different building blocks initiated in lattice assumption in Section 6.1.1

[6] There are other post-quantum assumptions like group action which one can base on and reduce the number of message flows at the cost of heavier computation, and there exists (s)aPAKE compilers that supports such assumption[110]. Our general compiler also supports such assumption, although we don't explicitly compare here

6.2 Compiler from PAKE to asymmetric PAKE

In Figure 6.3 we show our aPAKE compiler which transforms any UC PAKE protocol into a UC asymmetric PAKE (aPAKE) protocol, whose only additional cost is client-to-server authentication implemented using CCA-secure KEM.

Ideal PAKE Functionality. Our aPAKE protocol assumes a *revised* PAKE functionality $\mathcal{F}_{\mathsf{rpwKE}}$, which in addition to a session key outputs a protocol transcript to each party. Functionality $\mathcal{F}_{\mathsf{rpwKE}}$ was introduced by [72], since their Ω-method construction needs, just like our protocol, a cryptographic handle on the PAKE protocol instance. We include this revised functionality in Figure 2.1. (Our version of $\mathcal{F}_{\mathsf{rpwKE}}$ is slightly streamlined compared to the original in [72], but the differences are only syntactic.)

Note that the transcripts on any two sessions output by $\mathcal{F}_{\mathsf{rpwKE}}$ are forced to be different unless these two sessions are passively connected (i.e. both end up in fresh state after symmetric PAKE completes). Similarly to [72], the reason of this augmentation of $\mathcal{F}_{\mathsf{rpwKE}}$ is to make the key confirmation message τ non-malleable. Note that a man-in-the-middle adversary who compromises the server's password file can run the symmetric PAKE subprotocol with both the client and the server using the password hash found in that file. Since the UC PAKE functionality allows the attacker with a matching password to set the PAKE session key arbitrarily, the PAKE key can be the same on these two sessions, and the attacker could win by just forwarding any further authentication messages between the client to server. Adding a transcript to PAKE outputs, and requiring that no two PAKE instances can have the same transcript, and binding the authentication mechanism to that transcript, allows us to foil the above attack.

Theorem 6.1. *Protocol* APAKEM*, shown in Figure 6.3, realizes UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *in the* $\mathcal{F}_{\mathsf{rpwKE}}$*-hybrid world (i.e. assuming a realization of the* revised *UC PAKE functionality* $\mathcal{F}_{\mathsf{rpwKE}}$*) in the Random Oracle Model (ROM) for hash function* H*, assuming*

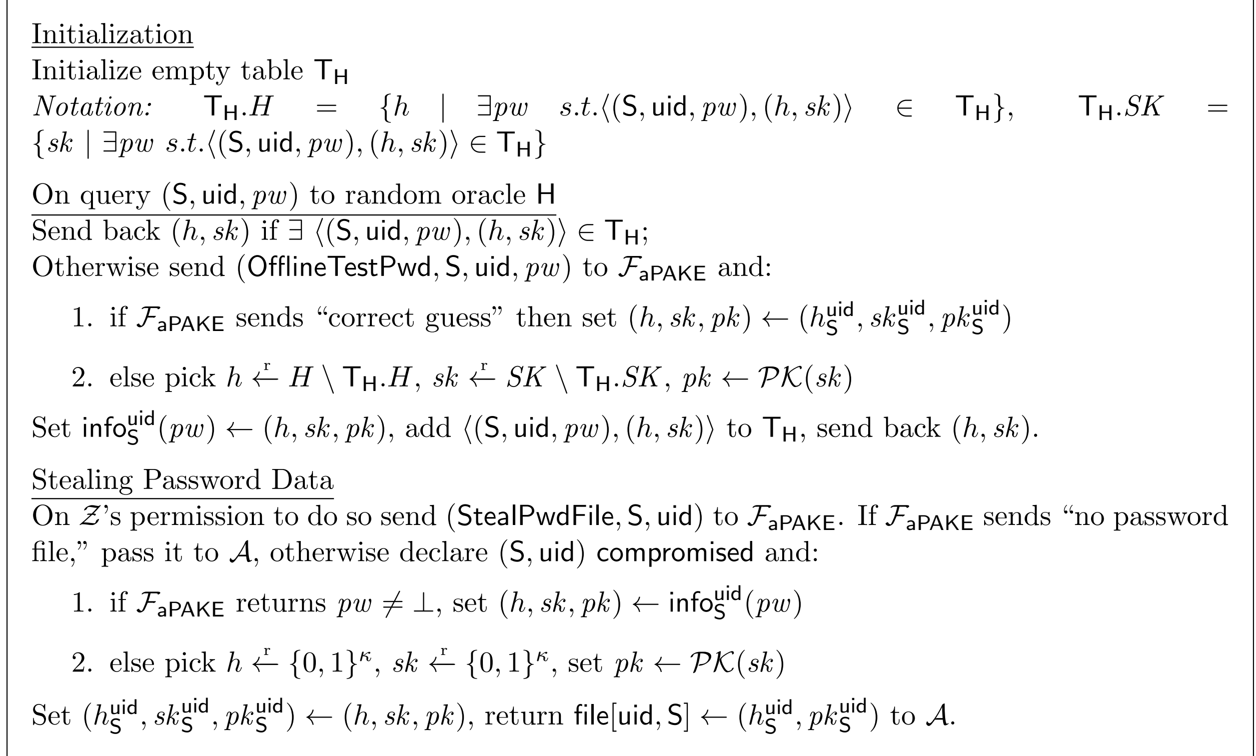
that KEM *is a CCA-secure KEM,* AE *is a CCA-secure, random-key robust, and unforgeable authenticated encryption scheme,* MAC *is an unforgeable and tag-random MAC, and* PRF *is a PRF.*

To prove Theorem 6.1 we show that the environment's view of the real-world security game, which is an interaction between the real-world adversary, the honest parties who follow the protocol, and functionality $\mathcal{F}_{\mathsf{rpwKE}}$ shown in Figure 2.1, is indistinguishable from the environment's view of the ideal-world, which is an interaction between simulator SIM in Figures 6.4 and 6.10, and functionality $\mathcal{F}_{\mathsf{aPAKE}}$. The UC aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$, taken from [72], is included for reference in Figure 2.4, along with functionality $\mathcal{F}_{\mathsf{rpwKE}}$ in Figure 2.1.

Simulator construction. We show an overview of our simulation strategy in Fig 6.5, which compares the real world execution with the ideal world one which involves the simulator SIM shown in Figures 6.46.10. SIM is split into two parts: Figure 6.4 contains the SIM pt.1 part of the diagram in Fig 6.5, i.e. it deals with adversarial hash queries, and in addition with the compromise of password files. Fig 6.10 contains the SIM pt.2 part of the diagram in Fig 6.5 dealing with online aPAKE sessions. In addition it also emulates functionality $\mathcal{F}_{\mathsf{rpwKE}}$.

Notation. In the simulator, for clear arguments we use $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}})$ to mark one session's status. We use hbc to mark a session's status as honest-but-curious and not attacked yet. We use act to mark the session as being actively attacked by $\mathcal{A}$, and we further mark it act(1) if adversary guessed the correct password, and act(2) if adversary has compromised the password file. In the simulator and games after Game 6, this act flag is also expanded to store honest server's pk as $\mathsf{act}(pk, b)$ to simplify the proof. We use rnd to denote all other cases.

In the proof we use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i. Hence the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G8}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let q_{pw}, q_{H} and q_{ses} be the upper-bounds on the number of resp. password files, H queries and



Initialization
Initialize empty table $\mathsf{T_H}$
Notation: $\mathsf{T_H}.H = \{h \mid \exists pw \;\; s.t.\langle(\mathsf{S}, \mathsf{uid}, pw), (h, sk)\rangle \in \mathsf{T_H}\}$, $\mathsf{T_H}.SK = \{sk \mid \exists pw \; s.t.\langle(\mathsf{S}, \mathsf{uid}, pw), (h, sk)\rangle \in \mathsf{T_H}\}$

On query $(\mathsf{S}, \mathsf{uid}, pw)$ to random oracle H
Send back (h, sk) if $\exists \; \langle(\mathsf{S}, \mathsf{uid}, pw), (h, sk)\rangle \in \mathsf{T_H}$;
Otherwise send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and:

1. if $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" then set $(h, sk, pk) \leftarrow (h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$
2. else pick $h \xleftarrow{\text{r}} H \setminus \mathsf{T_H}.H$, $sk \xleftarrow{\text{r}} SK \setminus \mathsf{T_H}.SK$, $pk \leftarrow \mathcal{PK}(sk)$

Set $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (h, sk, pk)$, add $\langle(\mathsf{S}, \mathsf{uid}, pw), (h, sk)\rangle$ to $\mathsf{T_H}$, send back (h, sk).

Stealing Password Data
On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$. If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "no password file," pass it to $\mathcal{A}$, otherwise declare $(\mathsf{S}, \mathsf{uid})$ compromised and:

1. if $\mathcal{F}_{\mathsf{aPAKE}}$ returns $pw \neq \perp$, set $(h, sk, pk) \leftarrow \mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw)$
2. else pick $h \xleftarrow{\text{r}} \{0,1\}^{\kappa}$, $sk \xleftarrow{\text{r}} \{0,1\}^{\kappa}$, set $pk \leftarrow \mathcal{PK}(sk)$

Set $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (h, sk, pk)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ to $\mathcal{A}$.

Figure 6.4: Simulator SIM showing that protocol APAKEM realizes $\mathcal{F}_{\mathsf{aPAKE}}$:Part 1

online C or S aPAKE sessions. Notations $\varepsilon_{\mathsf{MAC.sec}}, \varepsilon_{\mathsf{MAC.rand}}, \varepsilon_{\mathsf{prf}}, \varepsilon_{RBST}, \varepsilon_{AUTH}$ and $\varepsilon_{\mathsf{AE.sec}}$ are adversarial advantage against MAC unforgeability, MAC tag randomness, pseudorandomness of prf, AE key robustness, AE unforgeabiity and AE CCA security, as defined in Section 2. Note that the password hash in UC aPAKE, as defined by functionality $\mathcal{F}_{\mathsf{aPAKE}}$, requires that an offline password test corresponds to a unique choice of $(\mathsf{S}, \mathsf{uid})$, and we enforce that by setting the RO hash as $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw)$ in protocol APAKEM in Figure 6.3. Also note that in the SIM w.l.o.g we assume that server side receives the PAKE session key and the corresponding transcript before client. And in the proof part, notion "we" refers to the SIM.

Below we include the full proof of Theorem 6.1.

Game 0 *(real world)*: The real-world game shown in Figure 6.6 is the real world view of executing the protocol of Figure 6.3.

Game 1 *(random $h_{\mathsf{S}}^{\mathsf{uid}}$ and $sk_{\mathsf{S}}^{\mathsf{uid}}$ in the password file)*: We change the StorePwdFile process-

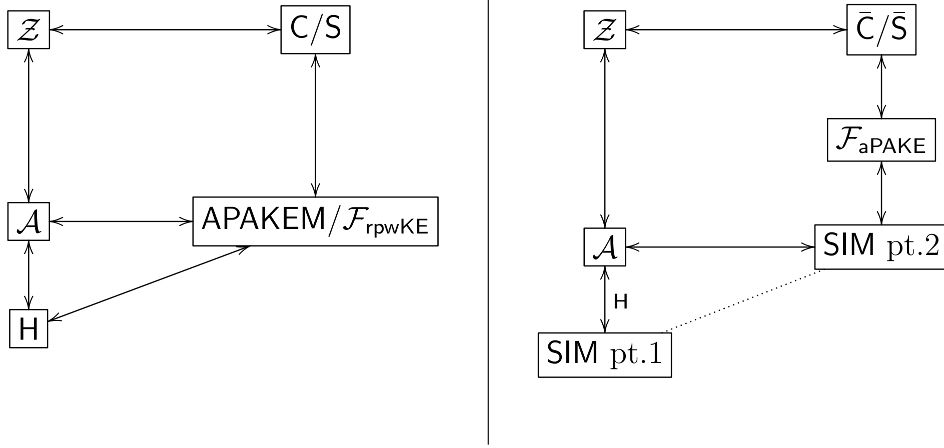


Figure 6.5: real-world (left) vs. simulation (right) for protocol APAKEM

ing part by picking $h_{\mathsf{S}}^{\mathsf{uid}}$ and $sk_{\mathsf{S}}^{\mathsf{uid}}$ as random elements in $\{0,1\}^{\kappa}$ and adding $\langle(\mathsf{S},\mathsf{uid},pw_{\mathsf{S}}^{\mathsf{uid}}),(h_{\mathsf{S}}^{\mathsf{uid}},sk_{\mathsf{S}}^{\mathsf{uid}})\rangle$ to T_{H}, instead of directly querying H. Then we pick corresponding $pk \leftarrow \mathcal{PK}(sk_{\mathsf{S}}^{\mathsf{uid}})$. If $h_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{T}_{\mathsf{H}}^{pw}.H$ or $sk_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{T}_{\mathsf{H}}^{pw}.SK$ for any pw, the game aborts. We further change CltSession where we directly retrieve $h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}$ instead of querying oracle H if client input $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$. This is only a syntactic change. The divergence introduced in this game is due to the probability of encountering abort, which leads to $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq q_{\mathsf{H}} q_{\mathsf{pw}}/2^{\kappa-1}$.

Game 2 *(emulate $\mathcal{F}_{\mathsf{rpwKE}}$)*: On $(\mathsf{TestPwd},\mathsf{sid}|\mathsf{C}|\mathsf{S},\mathsf{P},h)$ to $\mathcal{F}_{\mathsf{rpwKE}}$:

1. if $\mathcal{A}$ already stole the password file by sending $(\mathsf{StealPwdFile},\mathsf{S},\mathsf{uid})$, and $h = h_{\mathsf{S}}^{\mathsf{uid}}$, then we return "correct" and set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{act}(1)$

2. if there exists adversarial hash query $(\mathsf{S},\mathsf{uid},pw_{\mathsf{S}}^{\mathsf{uid}})$ to H, and $h = h_{\mathsf{S}}^{\mathsf{uid}}$, then we return "correct" and set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{act}(2)$

3. in any other case set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and return "incorrect"

We emulate TestPwd in $\mathcal{F}_{\mathsf{rpwKE}}$ as above. We output "correct" only in the following cases: (1) adversary stole password file, or (2) adversary compromised the password and queried the corresponding hash. The flag is only used internally, so this game looks identical to the previous one. We further emulate NewSession in $\mathcal{F}_{\mathsf{rpwKE}}$ by sending $(\mathsf{NewSession},\mathsf{sid}|\mathsf{C}|\mathsf{S},\mathsf{C},\mathsf{S})$

to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{rpwKE}}$, which is also a syntactic change. Since we no longer need to retrieve $h_{\mathsf{S}}^{\mathsf{uid}}$ in $\mathsf{SvrSession}$, or compute $h_{\mathsf{C}}^{\mathsf{sid}}$ in $\mathsf{CltSession}$, we remove the corresponding steps. The game is identical to the previous one from the environment's view, thus we have $|\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]|$.

GAME 3 *(abort if c valid under two different keys)*: In this game we bind the authenticated encryption ciphertext c to a single encryption key k_e (and therefore a single pw) by adding an abort if c is valid under two different keys.

Initialize empty table $\mathsf{T_H}$; (Notation $\mathsf{T_H}.H$ and $\mathsf{T_H}.SK$ as in Fig. 6.4)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Get $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow \mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$, set $pk_{\mathsf{S}}^{\mathsf{uid}} \leftarrow \mathcal{PK}(sk_{\mathsf{S}}^{\mathsf{uid}})$, and $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$
- On new $(\mathsf{S}, \mathsf{uid}, pw)$ to H: Pick $h \stackrel{\mathrm{r}}{\leftarrow} H \backslash \mathsf{T_H}.H$, $sk \stackrel{\mathrm{r}}{\leftarrow} SK \backslash \mathsf{T_H}.SK$, add $((\mathsf{S}, \mathsf{uid}, pw), (h, sk))$ to $\mathsf{T_H}$ and return (h, sk)
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Set $(h_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, \mathsf{C}, h_{\mathsf{S}}^{\mathsf{uid}})$ to $\mathcal{F}_{\mathsf{rpwKE}}$ and await for response
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, \mathsf{uid}, pw)$ to C: query $(\mathsf{S}, \mathsf{uid}, pw)$ to oracle H and obtain $(h_{\mathsf{C}}^{\mathsf{sid}}, sk_{\mathsf{C}}^{\mathsf{sid}})$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, \mathsf{S}, h_{\mathsf{C}}^{\mathsf{sid}})$ to $\mathcal{F}_{\mathsf{rpwKE}}$ and await for response
- S receives $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, K)$ and $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{transcript}, tr)$ from $\mathcal{F}_{\mathsf{rpwKE}}$: set $(e, k_m) \leftarrow \mathsf{enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$, $K' \leftarrow \mathsf{prf}(K, 0)$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $c \leftarrow \mathsf{AEnc}(k_e, e)$, send c to $\mathcal{Z}$
- C receives $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, K)$ and $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{transcript}, tr)$ from $\mathcal{F}_{\mathsf{rpwKE}}$, and c' from $\mathcal{Z}$: set $K' \leftarrow \mathsf{prf}(K, 0)$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $e \leftarrow \mathsf{ADec}(k_e, c')$ and
  1. if $e = \perp$ then output $\perp$
  2. else set $k_m \leftarrow \mathsf{dec}(sk_{\mathsf{C}}^{\mathsf{sid}}, e)$, $\tau \leftarrow \mathsf{Mac}(k_m, tr)$, send τ to $\mathcal{Z}$, output K'
- S receives τ' from $\mathcal{Z}$: If $\mathsf{Vrfy}(k_m, tr, \tau') = 1$ then output K', else output $\perp$

Figure 6.6: Game 0: $\mathcal{Z}$'s interaction with real-world protocol APAKEM

In passive sessions, once received an honest c from matching $\mathsf{S}^{\mathsf{sid}}$, $\mathsf{C}^{\mathsf{sid}}$ would output $\perp \leftarrow \mathsf{ADec}(k_e, c)$ when using a wrong key k_e, which further suggests that $\mathsf{C}^{\mathsf{sid}}$ is using $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. In Game 3 we abort on some $\mathsf{C}^{\mathsf{sid}}$-side event defined as Bad_1, where $\mathsf{C}^{\mathsf{sid}}$ uses a wrong k_e and

still decrypts this honest c to a message $m \neq \perp$, in which case Game 2 outputs τ, K' and Game 3 outputs $\perp$.

We construct a reduction $\mathcal{R}_1$ that reduces Bad_1 to the random-key robustness of symmetric authenticated encryption, where k' and k are the challenge keys: $\mathcal{R}_1$ sets k as the resulting key for ADec for $\mathsf{C}^{\mathsf{sid}}$ running on $pw_{\mathsf{S}}^{\mathsf{uid}}$ and sets k' as the resulting key for ADec for $\mathsf{C}^{\mathsf{sid}}$ running on $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. $\mathcal{R}_1$ runs the code of Game 2 except that it uses k' and k as input. In every client session $\mathcal{R}_1$ checks if neither $\mathsf{ADec}(k, c)$ or $\mathsf{ADec}(k', c)$ outputs $\perp$, and if so it outputs c and aborts the game. The probability of encountering this abort is bounded by $q_{\text{ses}} \cdot \varepsilon_{RBST}$.

Thus we have $|\Pr[\mathsf{G3}] - \Pr[\mathsf{G2}]| \leq q_{\text{ses}} \cdot \varepsilon_{RBST}$.

Game 4 *(abort if adversarial c^* valid without $\mathcal{A}$ computing k_e)*: In Game 4 we add an abort in the case that $\mathcal{A}$ replaces the KEM ciphertext c sent from $\mathsf{S}^{\mathsf{sid}}$ to $\mathsf{C}^{\mathsf{sid}}$ with $c^* \neq c$, and although $\mathcal{A}$ doesn't know k_e, $\mathsf{ADec}(k_e, c^*) \neq \perp$. In this case k_e is a random string from environment's view, and we can construct a reduction $\mathcal{R}$ to the unforgeability of AE in this case, where k_e is the challenge AE key. $\mathcal{R}$ encrypts $c \leftarrow \mathsf{AEnc}(k_e, e)$ using its encryption oracle, and in every client subsession $\mathcal{R}$ computes $\mathsf{ADec}(k_e, c^*)$ using decryption oracle. In each client subsession $\mathcal{R}$ checks if $c^* \neq c$ and $\mathsf{ADec}(k_e, c^*) \neq \perp$, if so $\mathcal{R}$ outputs c^* and solves AE unforgeability challenge. We have that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \varepsilon_{AUTH}$.

Game 5 *(random messages and keys in passive sessions)*: In this game, we change the passive sessions, where both $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ are fresh and messages are passively exchanged. We mark $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ as hbc at the beginning of $\mathsf{CltSession}$ and $\mathsf{SvrSession}$ commands, respectively. We first randomize c sent by $\mathsf{S}^{\mathsf{sid}}$, and when $\mathsf{C}^{\mathsf{sid}}$ receives this c we shortcut all processing and output τ and K' as random elements in $\{0,1\}^{\kappa}$. Furthermore, if $\mathsf{S}^{\mathsf{sid}}$ receives this τ then we shortcut the verification and set $\mathsf{S}^{\mathsf{sid}}$'s session key to K' output by $\mathsf{C}^{\mathsf{sid}}$. This change can be done in following steps:

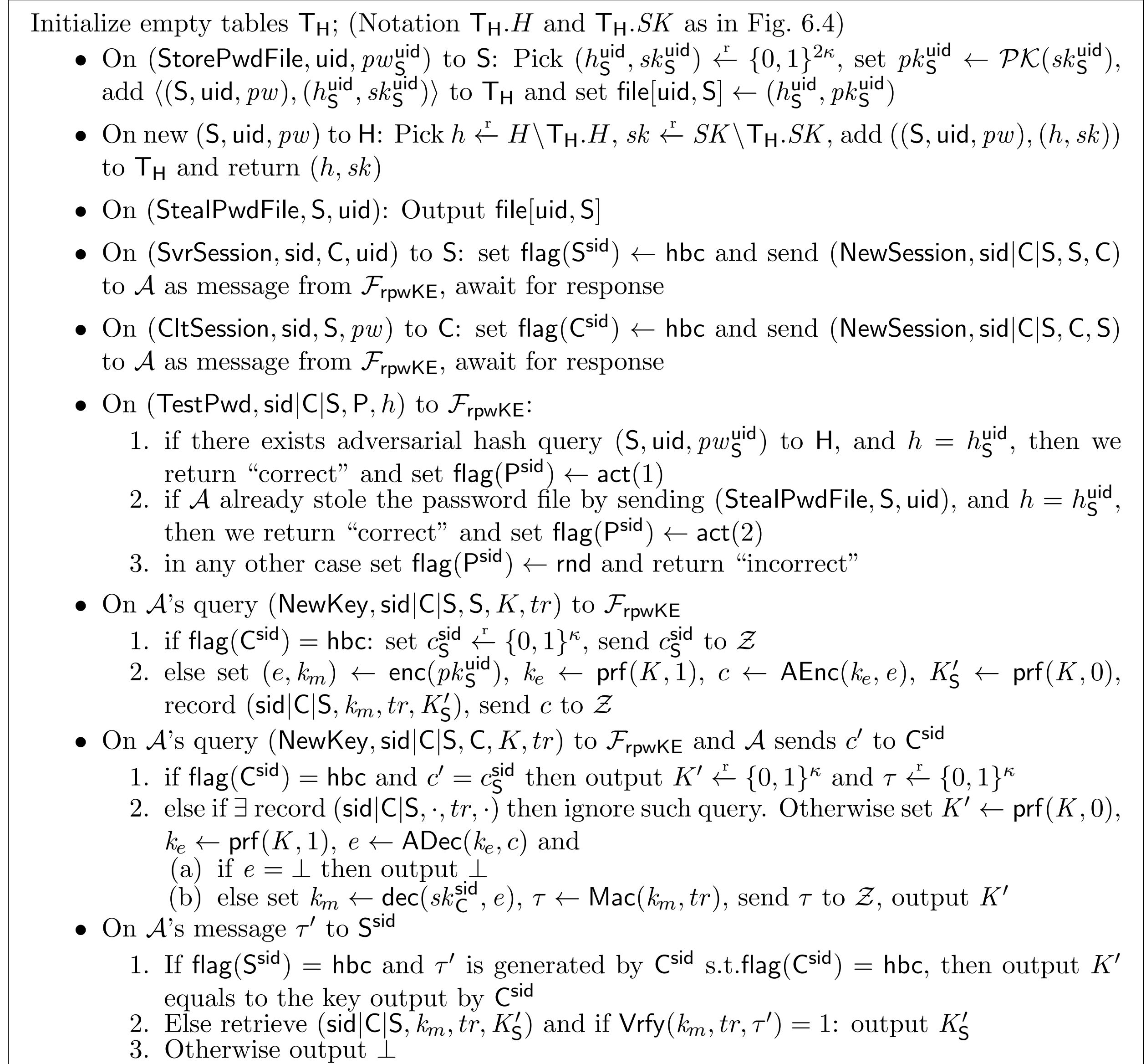
Step (a), recall that in Game 2 in passive sessions $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ receive same K and tr from $\mathcal{F}_{\mathsf{rpwKE}}$, and they further generate same $k_e \leftarrow \mathsf{prf}(K, 1)$ and $\mathsf{S}^{\mathsf{sid}}$ generates $c \leftarrow \mathsf{AEnc}(k_e, e)$ where $(e, k_m) \leftarrow \mathsf{Enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$, and in Game 3 we bind each ciphertext to a single k_e. In Game 5 we directly set $c \xleftarrow{r} \{0,1\}^\kappa$. This can be done in several sub-steps. First we randomize k_m for $\mathsf{S}^{\mathsf{sid}}$, and once $\mathsf{C}^{\mathsf{sid}}$ receives the c passively sent from the matching $\mathsf{S}^{\mathsf{sid}}$, we skip computing $e \leftarrow \mathsf{ADec}(k_e, c)$, $k_m \leftarrow \mathsf{Dec}(sk, e)$, and directly retrieve this k_m for $\mathsf{C}^{\mathsf{sid}}$, since in passive sessions $\mathsf{C}^{\mathsf{sid}}$ always gets same k_m as $\mathsf{S}^{\mathsf{sid}}$, this is only syntactic change. Second we randomize e instead of getting it from $\mathsf{Enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$, and send the ciphertext of e to $\mathsf{C}^{\mathsf{sid}}$ as before. This change introduces negligible difference under CPA security of AE, which can be proved under a hybrid argument over the number of encryption queries q_{AE}. In the i-th hybrid, $\mathcal{R}$ respond the first $i-1$ encryption queries of a random $e \leftarrow \{0,1\}^\kappa$, and respond the i-th to q_{AE}-th encryption of the real $e \leftarrow \mathsf{Enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$. The first hybrid is identical to Game 3 and the q_{AEnc}-th hybrid is identical to the current Game 4. If the environment can distinguish the i-th hybrid from the $i+1$-th hybrid, $\mathcal{R}$ picks a random index $j \leftarrow \{0, ..., q_{\mathrm{ses}}\}$, and embed the challenge ciphertext c^* into the i-th AE encryption query. If this encryption query doesn't occur in this j-th session then abort. The bit output by the environment indicating which game it sees will be returned as the solution to CPA security game. Thus this change is upper-bounded by $q_{\mathrm{ses}} \cdot q_{\mathrm{AE}} \cdot \varepsilon_{\mathsf{AE.sec}}$. Since e is now random, we further skip AEnc and directly get $c \leftarrow \{0,1\}^\kappa$.

Step(b), instead of letting client set $\tau \leftarrow \mathsf{Mac}(k_m, tr)$ and server verify, we directly set $\tau \xleftarrow{r} \{0,1\}^\kappa$ for $\mathsf{C}^{\mathsf{sid}}$, and skip the verification for $\mathsf{S}^{\mathsf{sid}}$ receiving this τ. This change on τ introduces negligible difference since the verification always passes in all previous games. This change on τ can be reduced to the tag-randomness of Mac (Definition 2.13) and therefore is bounded by $q_{\mathrm{ses}} \cdot \varepsilon_{\mathsf{MAC.rand}}$.

Step(c), we remove the usage of prf and directly generate session key K' as same random string for $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$. Because such sessions always compute same session keys in previous games, the change is negligible and can be reduced to the security of prf, which is bounded

by $q_{\text{ses}} \cdot \varepsilon_{\mathsf{prf}}$.

Now that in passive sessions (c, τ, K') are all random strings independent of e, k_m and $pk_{\mathsf{S}}^{\mathsf{uid}}$, we further remove the usage of $e, k_m, pk_{\mathsf{S}}^{\mathsf{uid}}$, including processing $(e, k_m) \leftarrow \mathsf{Enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$. To sum up, $|\Pr[\mathsf{G5}] - \Pr[\mathsf{G4}]| \leq q_{\text{ses}} \cdot (\varepsilon_{\mathsf{prf}} + \varepsilon_{\mathsf{MAC.rand}} + q_{\text{AE}} \cdot \varepsilon_{\mathsf{AE.sec}})$.



Initialize empty tables $\mathsf{T_H}$; (Notation $\mathsf{T_H}.H$ and $\mathsf{T_H}.SK$ as in Fig. 6.4)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Pick $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}) \xleftarrow{r} \{0,1\}^{2\kappa}$, set $pk_{\mathsf{S}}^{\mathsf{uid}} \leftarrow \mathcal{PK}(sk_{\mathsf{S}}^{\mathsf{uid}})$, add $\langle(\mathsf{S}, \mathsf{uid}, pw), (h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}})\rangle$ to $\mathsf{T_H}$ and set $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$
- On new $(\mathsf{S}, \mathsf{uid}, pw)$ to H: Pick $h \xleftarrow{r} H \backslash \mathsf{T_H}.H$, $sk \xleftarrow{r} SK \backslash \mathsf{T_H}.SK$, add $((\mathsf{S}, \mathsf{uid}, pw), (h, sk))$ to $\mathsf{T_H}$ and return (h, sk)
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$ and send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, \mathsf{C})$ to $\mathcal{A}$ as message from $\mathcal{F}_{\mathsf{rpwKE}}$, await for response
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ to C: set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$ and send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as message from $\mathcal{F}_{\mathsf{rpwKE}}$, await for response
- On $(\mathsf{TestPwd}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{P}, h)$ to $\mathcal{F}_{\mathsf{rpwKE}}$:
  1. if there exists adversarial hash query $(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to H, and $h = h_{\mathsf{S}}^{\mathsf{uid}}$, then we return "correct" and set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{act}(1)$
  2. if $\mathcal{A}$ already stole the password file by sending $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, and $h = h_{\mathsf{S}}^{\mathsf{uid}}$, then we return "correct" and set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{act}(2)$
  3. in any other case set $\mathsf{flag}(\mathsf{P}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and return "incorrect"
- On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, K, tr)$ to $\mathcal{F}_{\mathsf{rpwKE}}$
  1. if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}$: set $c_{\mathsf{S}}^{\mathsf{sid}} \xleftarrow{r} \{0,1\}^{\kappa}$, send $c_{\mathsf{S}}^{\mathsf{sid}}$ to $\mathcal{Z}$
  2. else set $(e, k_m) \leftarrow \mathsf{enc}(pk_{\mathsf{S}}^{\mathsf{uid}})$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $c \leftarrow \mathsf{AEnc}(k_e, e)$, $K'_{\mathsf{S}} \leftarrow \mathsf{prf}(K, 0)$, record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K'_{\mathsf{S}})$, send c to $\mathcal{Z}$
- On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, K, tr)$ to $\mathcal{F}_{\mathsf{rpwKE}}$ and $\mathcal{A}$ sends c' to $\mathsf{C}^{\mathsf{sid}}$
  1. if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}$ and $c' = c_{\mathsf{S}}^{\mathsf{sid}}$ then output $K' \xleftarrow{r} \{0,1\}^{\kappa}$ and $\tau \xleftarrow{r} \{0,1\}^{\kappa}$
  2. else if $\exists$ record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, \cdot, tr, \cdot)$ then ignore such query. Otherwise set $K' \leftarrow \mathsf{prf}(K, 0)$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $e \leftarrow \mathsf{ADec}(k_e, c)$ and
     (a) if $e = \perp$ then output $\perp$
     (b) else set $k_m \leftarrow \mathsf{dec}(sk_{\mathsf{C}}^{\mathsf{sid}}, e)$, $\tau \leftarrow \mathsf{Mac}(k_m, tr)$, send τ to $\mathcal{Z}$, output K'
- On $\mathcal{A}$'s message τ' to $\mathsf{S}^{\mathsf{sid}}$
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' is generated by $\mathsf{C}^{\mathsf{sid}}$ s.t.$\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}$, then output K' equals to the key output by $\mathsf{C}^{\mathsf{sid}}$
  2. Else retrieve $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K'_{\mathsf{S}})$ and if $\mathsf{Vrfy}(k_m, tr, \tau') = 1$: output K'_{S}
  3. Otherwise output $\perp$

Figure 6.7: $\mathcal{Z}$'s view after Game 5

GAME 6 *(random messages and keys in other sessions)*: In Game 5 we shortcut the processing of passive sessions, as shown in Figure 6.7. In Game 6 we change how $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ react in all other cases. Recall that all sessions marked as $\mathsf{act}(1)$ means $\mathcal{A}$ has compromised

the corresponding password file and $\mathsf{act}(2)$ means $\mathcal{A}$ has guessed the correct password. In both cases $\mathcal{A}$ knows the PAKE session key K and therefore k_e.

Upon receiving NewKey command, actively attacked $\mathsf{S}^{\mathsf{sid}}$ (marked as act) act as in the real world, i.e. generates $e, k_m, k_e, c, K'_{\mathsf{S}}$ and record $(\mathsf{sid}, k_m, tr, K'_{\mathsf{S}})$. In other cases where $\mathsf{S}^{\mathsf{sid}}$ marked rnd (meaning that $\mathcal{A}$ interfered with the PAKE session and receives a wrong session key K and k_e, and client side decryption always fails, which is same as in previous games), we just randomize c by following the procedures as step (a) in Game 4, and by the same reduction to CPA security of AE the difference introduced by this change is bounded by $q_{\text{ses}} \cdot q_{\text{AE}} \cdot \varepsilon_{\mathsf{AE.sec}}$.

We also change how $\mathsf{S}^{\mathsf{sid}}$ reacts for all other non-passive τ' cases: if τ' verification fails, we output $\perp$ as before; if τ' is verified, then we process as real world and output the correct K', except when $\mathcal{A}$ steals the password file without successfully guessing $pw_{\mathsf{S}}^{\mathsf{uid}}$, i.e. $\mathcal{A}$ hasn't queried $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$, but still send this verified τ' by successfully decrypting the KEM ciphertext. We abort in this case, denoted as event Bad, and we argue that Bad can only happen with negligible probability, and can be reduced to an attack on CCA security of KEM. We construct such a reduction called $\mathcal{R}$, which works as follows: on KEM's CCA-security challenge (pk^*, e^*, k_m^*), $\mathcal{R}$ picks an index $i \xleftarrow{r} \{1, 2, ..., q_{\text{ses}}\}$ and embeds pk^* as $pk_{\mathsf{S}}^{\mathsf{uid}}$ for the i-th session from all sever sessions indexed from $\{1, 2, ..., q_{\text{ses}}\}$. $\mathcal{R}$ also picks an index $j \xleftarrow{r} \{1, 2, ..., q_{\text{ses}}\}$, and embeds (e^*, k_m^*) as the KEM encryption of this j-th session outputs. e^* is then encrypted and sent to $\mathcal{A}$. After $\mathcal{A}$ finishes all the processing and sends back τ^* in the j-th session, $\mathcal{R}$ immediately breaks CCA-security once it sees τ^* passes verification and server outputs a key instead of $\perp$.

We also add an abort if $\mathcal{A}$ successfully forges a valid Mac tag $\tau' \leftarrow \mathsf{Mac}(k_m, tr')$ after seeing $\tau \leftarrow \mathsf{Mac}(k_m, tr)$ output by honest client, where $tr \neq tr'$. This abort happens with negligible probability and can be reduced to MAC security, thus is bounded by $q_{\text{ses}}\varepsilon_{\mathsf{MAC.sec}}$.

We change how $\mathsf{C}^{\mathsf{sid}}$ reacts for all c' cases other than in fresh sessions in Game 5. For actively attacked sessions, Game 6 acts in the same way as in Game 5. In any other case, i.e. for $\mathsf{C}^{\mathsf{sid}}$ marked rnd and received a random K as PAKE session key, we shortcut all the process and output $\perp$ since the probability of a successful ADec decryption in this case is already aborted in Game 3. Thus we have $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}} \cdot (\varepsilon_{\mathsf{KEM.sec}} + \varepsilon_{\mathsf{MAC.sec}} + q_{\text{AE}} \cdot \varepsilon_{\mathsf{AE.sec}})$.

Game 7 *(delay $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ generation until password compromise)*: In Game 6 $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ are initialized in $\mathsf{StorePwdFile}$, in Game 7 we postpone these steps until password compromise. This change can be done in several steps.

Step(a), we delay generating $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ in $\mathsf{StorePwdFile}$ to $\mathsf{StealPwdFile}$. This introduce no difference since $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ is not used anywhere else.

Step(b), we remove steps of initiating $h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}}$ in $\mathsf{StorePwdFile}$ and instead delay them to $\mathsf{StealPwdFile}$ or $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$, depending on which happens first. In order to set $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ only after $\mathcal{A}$ finds $pw_{\mathsf{S}}^{\mathsf{uid}}$ via a successful offline dictionary attack, we mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ $\mathsf{compromised}$ anytime when $\mathcal{A}$ runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$. (case 1) If $\mathcal{A}$ first runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, we pick $h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}}$ at random, and later upon query $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$, if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is already marked $\mathsf{compromised}$, we simply retrieve $h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}}$, store them into $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ file and add to $\mathsf{T_H}$. (case 2) If $\mathcal{A}$ runs $\mathsf{H}(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ first, which means at this moment $pw_{\mathsf{S}}^{\mathsf{uid}}$ must be fresh, we treat it the same way as before, just like other $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. We also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, and later if $\mathcal{A}$ runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ and record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$ exists, we directly retrieve $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ from $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ file and set them to the corresponding password file. We also simultaneously change the corresponding part in $\mathsf{TestPwd}$, where we replace "if there exists adversarial hash query $(\mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to H" with checking "if $\exists pw$ s.t. $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (h, sk, pk)$", and replace "if $\mathcal{A}$ already stole the password file by sending $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$" with checking "if $\exists (h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ s.t. $h_{\mathsf{S}}^{\mathsf{uid}} = h$", which is only syntactic changes. We further treat $\mathsf{TestPwd}$ into client and server

case separately, and mark flag with additional internal info. Game 7(b) is identical to Game 7(a) since we only postpone $(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ generation. Thus $|\Pr[\mathsf{G7}] = \Pr[\mathsf{G6}]|$.

GAME 8 *(ideal world)*: This is the ideal-world interaction, i.e. an interaction of environment $\mathcal{Z}$ with simulator SIM and functionality $\mathcal{F}_{\mathsf{aPAKE}}$, shown in Figure 6.11. Observe that Game 7 is identical to the ideal world Game 8, which completes our argument that the real-world and the ideal-world interaction are indistinguishable to $\mathcal{Z}$, and thus completes the proof of Theorem 6.1.

KG-oblivious variant of our aPAKE construction. W also show a key-generation oblivious variant of our general PAKE-to-aPAKE compiler in Figure 6.8. Similar to GMR compiler, the client doesn't directly derive KEM's secret key from hash on password, instead server stores the secret key encrypted under password, and send to client during online sessions who later decrypts with the correct password. This variant reduces the computation cost on server side during online phase, meanwhile it requires extra storage, and the main reason we show this variant is to further increase the generality of the compiler, such that it can fit those cryptographic schemes whose secret keys are hard to derive from hash, such as RSA signatures/KEMs.

6.3 An Efficient Instantiation of Our Compiler

To concretely show the practicality of our PAKE-to-aPAKE compiler, in this section we present an efficient instantiation of our aPAKE protocol in Figure 6.9, with the PAKE subprotocol instantiated with the UC PAKE from [67]. The Randomized Ideal Cipher (RIC) can be realized by the modified 2-round Feistel(m2F). The concrete instantiations of other building blocks such as MAC and AE are discussed in Section 6.1.2. Since the underlying PAKE subprotocol is two round, we obtain a highly efficient UC asymmetric PAKE, with

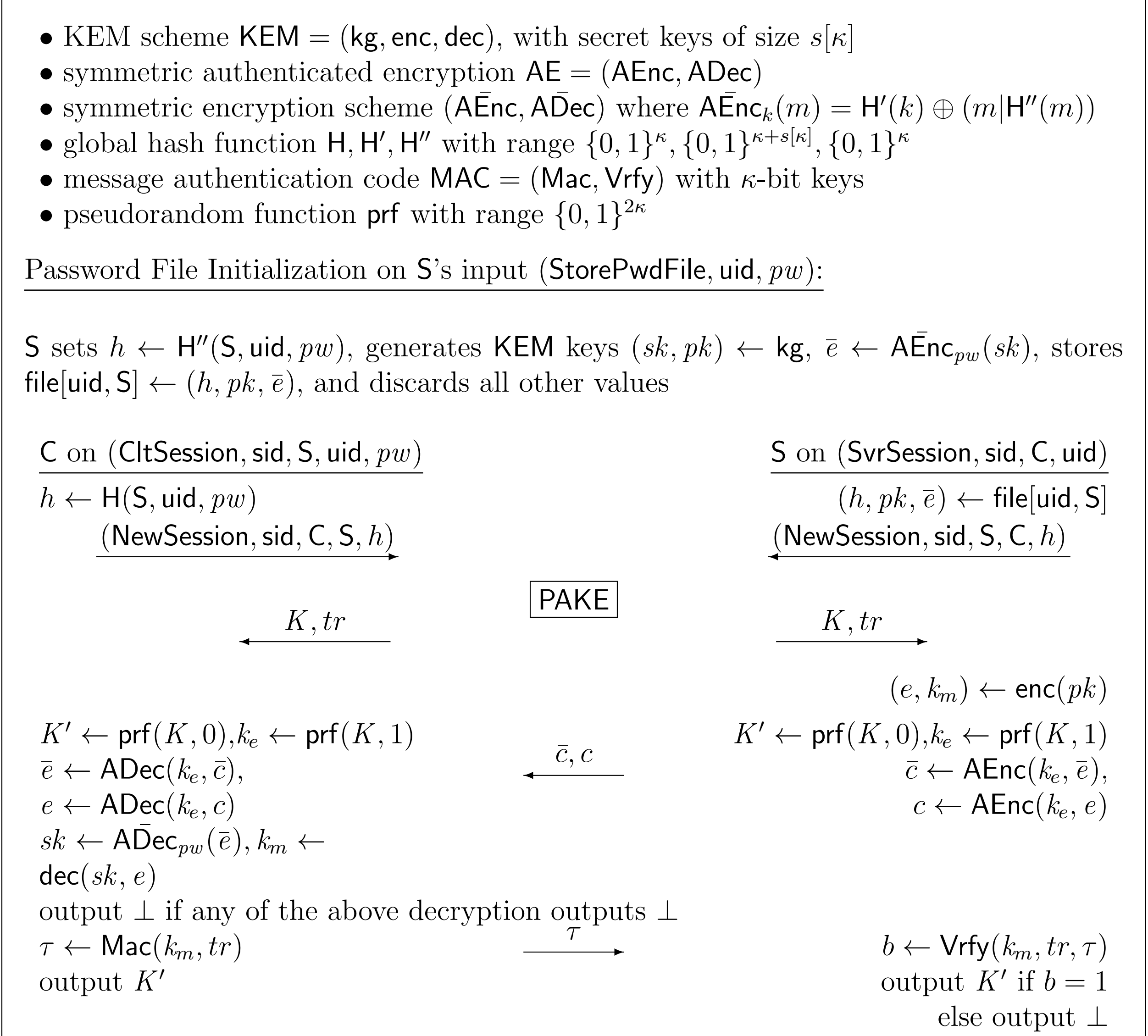


Figure 6.8: Key-Generation Oblivious variant of our PAKE-to-aPAKE compiler

only 3 rounds of communications (i.e. 3 message flows).

[67] provides a two-round PAKE protocol called EKE-KEM that can be built upon KEM which only requires weak anonymity property (Definition 2.6).

Note that this weak anonymity property is satisfied by many existing lattice KEMs, including the ones competing in NIST standardization, e.g. Kyber and Saber, whose key exchange protocol can be seen as the plain KEM which is CPA-secure and proved to be weak anonymous in [67]. Since these lattice KEMs can also derive their CCA-secure versions

via Fujisaki-Okamoto transform, our concrete instantiation also enjoys a favor in real world implementation, as KEM_0 can be realized by the plain KEM, and KEM_1 can be just the corresponding CCA-secure version of KEM_0, and usually both versions are supported and implemented in the same code library of these lattice based KEMs.

We also want to point out that this instantiation itself can also be seen as a KEM-to-aPAKE compiler. Thus our KEM-to-aPAKE compiler is the first aPAKE compiler that solely based on KEM, and can be efficiently built upon lattice[7].

[7] [84] also provides a KEM-to-aPAKE compiler but as discussed in Section 6.1.1 it cannot be generalized to base on lattice assumptions, due to technical difficulties in constructing a lattice based KEM satisfying both OW-PCA and strong anonymity properties (Definition 2.7).

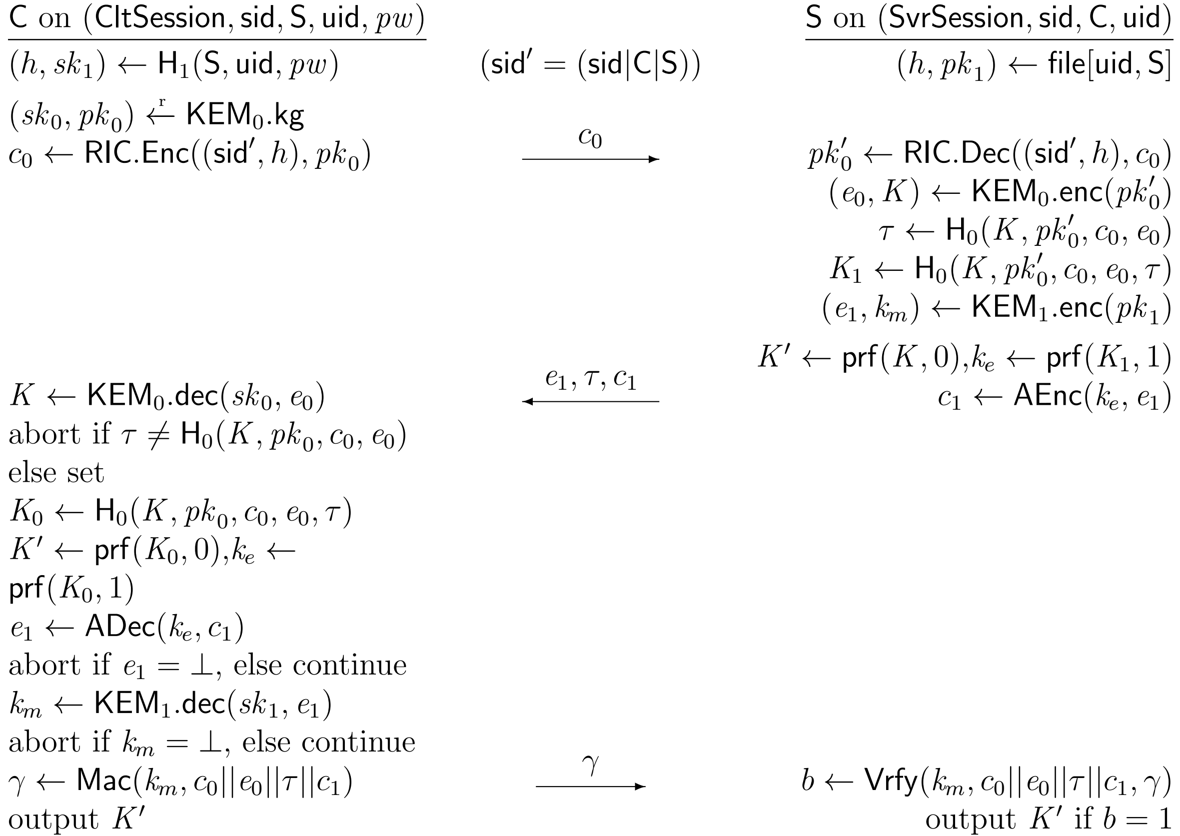


Figure 6.9: A three-round UC asymmetric PAKE using compiler APAKEM instantiated with UC PAKE protocol from [67]

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$

set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as $\mathcal{F}_{\mathsf{rpwKE}}$'s msg and await for response.

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$

set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, \mathsf{C})$ to $\mathcal{A}$ as $\mathcal{F}_{\mathsf{rpwKE}}$'s msg and await for response.

On $\mathcal{A}$'s query $(\mathsf{TestPwd}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, h)$ to $\mathcal{F}_{\mathsf{rpwKE}}$ (only respond to first such query)

1. If $\exists pw$ s.t. $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (h, sk, pk)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, if return "correct guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}(sk, 1)$, return "correct"
2. Else if $\exists(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ s.t. $h_{\mathsf{S}}^{\mathsf{uid}} = h$ then send $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$, if return "correct guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}(sk_{\mathsf{S}}^{\mathsf{uid}}, 2)$, return "correct"
3. In any other case set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, return "incorrect"

On $\mathcal{A}$'s query $(\mathsf{TestPwd}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, h)$ to $\mathcal{F}_{\mathsf{rpwKE}}$ (only respond to first such query)

1. If $\exists pw$ s.t. $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (h, sk, pk)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and if $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess" then set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}(pk, 1)$, return "correct"
2. Else if S is $\mathsf{compromised}$ and $\exists(h_{\mathsf{S}}^{\mathsf{uid}}, sk_{\mathsf{S}}^{\mathsf{uid}}, pk_{\mathsf{S}}^{\mathsf{uid}})$ s.t. $h = h_{\mathsf{S}}^{\mathsf{uid}}$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}(pk_{\mathsf{S}}^{\mathsf{uid}}, 2)$, return "correct"
3. In any other case set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and return "incorrect"

On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, K, tr)$ to $\mathcal{F}_{\mathsf{rpwKE}}$

1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ then output $c_{\mathsf{S}}^{\mathsf{sid}} \xleftarrow{\mathrm{r}} \{0,1\}^{\kappa}$
2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}(pk, \cdot)$, compute $(e, k_m) \leftarrow \mathsf{enc}(pk)$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $c \leftarrow \mathsf{AEnc}(k_e, e)$, and output c; record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K)$
3. In any other case, output $c \xleftarrow{\mathrm{r}} \{0,1\}^{\kappa}$

On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, K, tr)$ to $\mathcal{F}_{\mathsf{rpwKE}}$ and $\mathcal{A}$ sends c' to $\mathsf{C}^{\mathsf{sid}}$

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}$ and $c' = c_{\mathsf{S}}^{\mathsf{sid}}$ generated by SIM for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and output $\tau \xleftarrow{\mathrm{r}} \{0,1\}^{\kappa}$
2. Else, if $\exists$ record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, \cdot, tr, \cdot)$ then ignore such NewKey query. Otherwise:
   (a) If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}(sk, b)$, set $k_e \leftarrow \mathsf{prf}(K, 1)$, $e \leftarrow \mathsf{ADec}(k_e, c')$, and
      i. if $e = \perp$ then output $\perp$
      ii. if $e \neq \perp$ then set $k_m \leftarrow \mathsf{dec}(sk, e)$, set $\tau \leftarrow \mathsf{Mac}(k_m, tr)$, and send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{prf}(K, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and output τ
   (b) In other cases, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp)$, $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, output $\perp$

On $\mathcal{A}$'s message τ' to $\mathsf{S}^{\mathsf{sid}}$

1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' is generated by SIM for $\mathsf{C}^{\mathsf{sid}}$ s.t.$\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}$, then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$
2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}(pk, b)$, retrieve $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K)$(if not exist go to 3.) and :
   (a) if $\mathsf{Vrfy}(k_m, tr, \tau') \neq 1$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$
   (b) if $\mathsf{Vrfy}(k_m, tr, \tau') = 1$:
      i. if $b = 1$: send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \mathsf{prf}(K, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$
      ii. if $b = 2$: if $\exists$ $\mathsf{info}_{\mathsf{S}}^{\mathsf{uid}}(pw) = (\cdot, \cdot, pk)$ then send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, pw)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \mathsf{prf}(K, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$; else abort.
3. Else send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, \perp)$, $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$

Figure 6.10: Simulator SIM showing that protocol APAKEM realizes $\mathcal{F}_{\mathsf{aPAKE}}$:Part 2

Initialize empty table $\mathsf{T_H}$; *Notation:* $\mathsf{T_H}.H, \mathsf{T_H}.SK$ from Figure 6.4

- On query $(\mathsf{S}, \mathsf{uid}, pw)$ to random oracle H
  1. If $pw_\mathsf{S}^\mathsf{uid}$ is $\mathsf{compromised}$ and $pw = pw_\mathsf{S}^\mathsf{uid}$: set $(h, sk, pk) \leftarrow (h_\mathsf{S}^\mathsf{uid}, sk_\mathsf{S}^\mathsf{uid}, pk_\mathsf{S}^\mathsf{uid})$
  2. If $pw_\mathsf{S}^\mathsf{uid}$ is fresh or $pw \neq pw_\mathsf{S}^\mathsf{uid}$, record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, pick $h \xleftarrow{r} H \setminus \mathsf{T_H}.H$, $sk \xleftarrow{r} SK \setminus \mathsf{T_H}.SK$, $pk \leftarrow \mathcal{PK}(sk)$

  In both cases set $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) \leftarrow (h, sk, pk)$, add $\langle (\mathsf{S}, \mathsf{uid}, pw), (h, sk) \rangle$ to $\mathsf{T_H}$, output (h, sk).
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: mark $pw_\mathsf{S}^\mathsf{uid}$$\mathsf{compromised}$ and
  1. If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_\mathsf{S}^\mathsf{uid} \rangle$ then set $(h, sk, pk) \leftarrow \mathsf{info}_\mathsf{S}^\mathsf{uid}(pw)$
  2. Else pick $h \xleftarrow{r} \{0,1\}^\kappa$, $sk \xleftarrow{r} \{0,1\}^\kappa$, set $pk \leftarrow \mathcal{PK}(sk)$

  In either case set $(h_\mathsf{S}^\mathsf{uid}, sk_\mathsf{S}^\mathsf{uid}, pk_\mathsf{S}^\mathsf{uid}) \leftarrow (h, sk, pk)$, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (h_\mathsf{S}^\mathsf{uid}, pk_\mathsf{S}^\mathsf{uid})$.
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_\mathsf{aPAKE}$: set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{hbc}$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ as message from $\mathcal{F}_\mathsf{rpwKE}$, await for response
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_\mathsf{aPAKE}$: set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{hbc}$, send $(\mathsf{NewSession}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, \mathsf{C})$ to $\mathcal{A}$ as message from $\mathcal{F}_\mathsf{rpwKE}$, await for response
- On $\mathcal{A}$'s query $(\mathsf{TestPwd}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, h)$ to $\mathcal{F}_\mathsf{rpwKE}$:(only respond to first such message)
  1. If $\exists pw$ s.t. $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) = (h, sk, pk)$ and $pw = pw_\mathsf{S}^\mathsf{uid}$ then set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{act}_\mathsf{S}^\mathsf{uid}(sk, 1)$ and return "correct"
  2. Else if $\exists (h_\mathsf{S}^\mathsf{uid}, sk_\mathsf{S}^\mathsf{uid}, pk_\mathsf{S}^\mathsf{uid})$ s.t. $h_\mathsf{S}^\mathsf{uid} = h$: set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{act}_\mathsf{S}^\mathsf{uid}(sk_\mathsf{S}^\mathsf{uid}, 2)$, return "correct"
  3. In any other case set $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) \leftarrow \mathsf{rnd}$ and return "incorrect"
- On $\mathcal{A}$'s query $(\mathsf{TestPwd}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, h)$ to $\mathcal{F}_\mathsf{rpwKE}$:(only respond to first such message)
  1. If $\exists pw$ s.t. $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) = (h, sk, pk)$ and $pw = pw_\mathsf{S}^\mathsf{uid}$, set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{act}(pk, 1)$ and return "correct"
  2. If $pw_\mathsf{S}^\mathsf{uid}$ is $\mathsf{compromised}$ and $\exists (h_\mathsf{S}^\mathsf{uid}, sk_\mathsf{S}^\mathsf{uid}.pk_\mathsf{S}^\mathsf{uid})$ s.t. $h = h_\mathsf{S}^\mathsf{uid}$: set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{act}(pk_\mathsf{S}^\mathsf{uid}, 2)$ and return "correct"
  3. In any other case set $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) \leftarrow \mathsf{rnd}$ and return "incorrect"
- On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{S}, K, tr)$ to $\mathcal{F}_\mathsf{rpwKE}$
  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$ then output $c_\mathsf{S}^\mathsf{sid} \xleftarrow{r} \{0,1\}^\kappa$
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}(pk, \cdot)$, compute $(e, k_m) \leftarrow \mathsf{enc}(pk)$, $k_e \leftarrow \mathsf{prf}(K, 1)$, $c \leftarrow \mathsf{AEnc}(k_e, e)$, and output c; record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K)$
  3. In any other case, output $c \xleftarrow{r} \{0,1\}^\kappa$
- On $\mathcal{A}$'s query $(\mathsf{NewKey}, \mathsf{sid}|\mathsf{C}|\mathsf{S}, \mathsf{C}, K, tr)$ to $\mathcal{F}_\mathsf{rpwKE}$ and $\mathcal{A}$ sends c' to C^sid
  1. If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}$ and $c' = c_\mathsf{S}^\mathsf{sid}$ then output $K' \xleftarrow{r} \{0,1\}^\kappa$ and $\tau \xleftarrow{r} \{0,1\}^\kappa$
  2. Else, if $\exists$ record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, \cdot, tr, \cdot)$ then ignore such query. Otherwise:
     (a) If $\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{act}_\mathsf{S}^\mathsf{uid}(sk, b)$, set $k_e \leftarrow \mathsf{prf}(K, 1)$, $e \leftarrow \mathsf{ADec}(k_e, c)$, and
        i. if $e = \perp$: output $\perp$
        ii. else set $k_m \leftarrow \mathsf{dec}(sk, e)$, output $\tau \leftarrow \mathsf{Mac}(k_m, tr)$ and $K' \leftarrow \mathsf{prf}(K, 0)$
     (b) In any other case output $\perp$
- On $\mathcal{A}$'s message τ' to S^sid
  1. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{hbc}$ and τ' is generated by C^sid s.t.$\mathsf{flag}(\mathsf{C}^\mathsf{sid}) = \mathsf{hbc}$, then output K' equals to the key output by C^sid
  2. If $\mathsf{flag}(\mathsf{S}^\mathsf{sid}) = \mathsf{act}(pk, b)$, retrieve record $(\mathsf{sid}|\mathsf{C}|\mathsf{S}, k_m, tr, K)$(if not exist goto 3.). If $\mathsf{Vrfy}(k_m, tr, \tau') = 1$ then:
     (a) if $b = 1$, then output $K' \leftarrow \mathsf{prf}(K, 0)$
     (b) if $b = 2$, then if $\exists$ $\mathsf{info}_\mathsf{S}^\mathsf{uid}(pw) = (\cdot, \cdot, pk)$ and $pw = pw_\mathsf{S}^\mathsf{uid}$ then output $K' \leftarrow \mathsf{prf}(K, 0)$; else abort.
  3. In any other case output $\perp$

Figure 6.11: Game 8: $\mathcal{Z}$'s interaction with ideal-world protocol APAKEM

Bibliography

[1] Facebook stored hundreds of millions of passwords in plain text, https://www.theverge.com/2019/3/21/18275837/facebook-plain-text-password-storage-hundreds-millions-users.

[2] Google stored some passwords in plain text for fourteen years, https://www.theverge.com/2019/5/21/18634842/google-passwords-plain-text-g-suite-fourteen-years.

[3] Nist round 3 selection result, https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, 2022.

[4] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In *Advances in Cryptology - CRYPTO 2020*, pages 278–307, 2020.

[5] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, Aug. 2020.

[6] M. Abdalla, M. Bellare, and P. Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In *Cryptographer's Track at RSA Conference, CT-RSA*, pages 143–158, 2001.

[7] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In *Topics in Cryptology – CT-RSA 2008*, pages 335–351. Springer, 2008.

[8] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In T. Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 335–351. Springer, Heidelberg, Apr. 2008.

[9] M. Abdalla, T. Eisenhofer, E. Kiltz, S. Kunzweiler, and D. Riepel. Password-authenticated key exchange from group actions. Cryptology ePrint Archive, Paper 2022/770, 2022. https://eprint.iacr.org/2022/770.

[10] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology – CT-RSA 2005*, pages 191–208. Springer, 2005.

[11] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology – CT-RSA 2005*, pages 191–208. Springer, 2005.

[12] B. Abdolmaleki, S. Badrinarayanan, R. Fernando, G. Malavolta, A. Rahimi, and A. Sahai. Two-round concurrent 2pc from sub-exponential lwe. Cryptology ePrint Archive, Paper 2022/1719, 2022. https://eprint.iacr.org/2022/1719.

[13] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang. Classic mceliece: Nist round 3 submission, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, 2021.

[14] M. R. Albrecht, A. Davidson, A. Deo, and D. Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and fhe. Cryptology ePrint Archive, Paper 2023/232, 2023. https://eprint.iacr.org/2023/232.

[15] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. Cryptology ePrint Archive, Paper 2019/1271, 2019. https://eprint.iacr.org/2019/1271.

[16] . E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebil. Frodokem: Nist round 3 submission, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, 2021.

[17] E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, and J. P. Steinberger. On the indifferentiability of key-alternating ciphers. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 531–550. Springer, Heidelberg, Aug. 2013.

[18] D. F. Aranha, P.-A. Fouque, C. Qian, M. Tibouchi, and J.-C. Zapalowicz. Binary elligator squared. In A. Joux and A. M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 20–37. Springer, Heidelberg, Aug. 2014.

[19] A. Arriaga, M. Barbosa, S. Jarecki, and M. Skrobot. C'est très CHIC: A compact password-authenticated key exchange from lattice-based kem. Cryptology ePrint Archive, Paper 2024/308, 2024.

[20] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 719–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[21] A. Basso. A post-quantum round-optimal oblivious prf from isogenies. Cryptology ePrint Archive, Paper 2023/225, 2023. https://eprint.iacr.org/2023/225.

[22] H. Beguinet, C. Chevalier, D. Pointcheval, T. Ricosset, and M. Rossi. Get a cake: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In *Applied Cryptography and Network Security, ACNS 2023, Kyoto, Japan*. Springer-Verlag, 2023.

[23] H. Beguinet, C. Chevalier, D. Pointcheval, T. Ricosset, and M. Rossi. Get a cake: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. Cryptology ePrint Archive, Paper 2023/470, 2023. `https://eprint.iacr.org/2023/470`.

[24] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key privacy in public-key encryption. In *Advances in Cryptology – ASIACRYPT 2001*. Springer, 2001.

[25] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, Dec. 2001.

[26] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, pages 139–155. Springer, 2000.

[27] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.

[28] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 1992*, pages 72–84. IEEE, 1992.

[29] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security – CCS 1993*, pages 244–250. ACM, 1993.

[30] F. Benhamouda, O. Blazy, L. Ducas, and W. Quach. Hash proof systems over lattices revisited. In *Public-Key Cryptography – PKC 2018*, volume 10770 of *Public-Key Cryptography – PKC 2018*, pages 644–674. Springer, 2018.

[31] F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013.

[32] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, Nov. 2013.

[33] D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, Y. Todo, and B. Viguier. Gimli : A cross-platform permutation. In W. Fischer and N. Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 299–320. Springer, Heidelberg, Sept. 2017.

[34] D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, Y. Todo, and B. Viguier. Gimli: a cross-platform permutation. Cryptology ePrint Archive, Report 2017/630, 2017. `http://eprint.iacr.org/2017/630`.

[35] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Heidelberg, May 2013.

[36] S. Bettaieb, L. Bidoux, O. Blazy, Y. Connan, and P. Gaborit. A gapless code-based hash proof system based on rqc and its applications. Cryptology ePrint Archive, Paper 2021/026, 2021. `https://eprint.iacr.org/2021/026`.

[37] W. Beullens, L. Dodgson, S. Faller, and J. Hesse. The 2hash oprf framework and efficient post-quantum instantiations. Cryptology ePrint Archive, Paper 2024/450, 2024. `https://eprint.iacr.org/2024/450`.

[38] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In B. Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 114–130. Springer, Heidelberg, Feb. 2002.

[39] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, Aug. 2002.

[40] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Paper 2017/634, 2017. `https://eprint.iacr.org/2017/634`.

[41] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – EUROCRYPT 2000*, pages 156–171. Springer, 2000.

[42] V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000.

[43] T. Bradley, J. Camenisch, S. Jarecki, A. Lehmann, G. Neven, and J. Xu. Password-authenticated public-key encryption. In *ACNS*, volume 11464 of *Lecture Notes in Computer Science*, pages 442–462. Springer, 2019.

[44] T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In *Advances in Cryptology - CRYPTO 2019*, pages 798–825, 2019.

[45] E. Brier, J.-S. Coron, T. Icart, D. Madore, H. Randriam, and M. Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, Aug. 2010.

[46] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[47] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science – FOCS 2001*, pages 136–145. IEEE, 2001.

[48] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT 2005*, pages 404–421. Springer, 2005.

[49] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, pages 453–474. Springer, 2001.

[50] J.-S. Coron, Y. Dodis, A. Mandal, and Y. Seurin. A domain extender for the ideal cipher. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 273–289. Springer, Heidelberg, Feb. 2010.

[51] J.-S. Coron, J. Patarin, and Y. Seurin. The random oracle model and the ideal cipher model are equivalent. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 1–20. Springer, Heidelberg, Aug. 2008.

[52] D. Dachman-Soled, J. Katz, and A. Thiruvengadam. 10-round Feistel is indifferentiable from an ideal cipher. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 649–678. Springer, Heidelberg, May 2016.

[53] J. Daemen, S. Hoffert, G. V. Assche, and R. V. Keer. The design of Xoodoo and Xoofff. *IACR Trans. Symm. Cryptol.*, 2018(4):1–38, 2018.

[54] Y. Dai, Y. Seurin, J. P. Steinberger, and A. Thiruvengadam. Indifferentiability of iterated Even-Mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 524–555. Springer, Heidelberg, Aug. 2017.

[55] Y. Dai and J. P. Steinberger. Indifferentiability of 8-round Feistel networks. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Heidelberg, Aug. 2016.

[56] J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In A. Joux, A. Nitaj, and T. Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2018*, pages 282–305, Cham, 2018. Springer International Publishing.

[57] A. Desai. The security of all-or-nothing encryption: Protecting against exhaustive key search. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 359–375. Springer, Heidelberg, Aug. 2000.

[58] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[59] J. Ding, S. Alsayigh, J. Lancrenon, S. RV, and M. Snook. Provably secure password authenticated key exchange based on rlwe for the post-quantum world. In H. Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, pages 183–204, Cham, 2017. Springer International Publishing.

[60] Y. Dodis and P. Puniya. Feistel networks made public, and applications. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 534–554. Springer, Heidelberg, May 2007.

[61] Y. Dodis, M. Stam, J. P. Steinberger, and T. Liu. Indifferentiability of confusion-diffusion networks. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 679–704. Springer, Heidelberg, May 2016.

[62] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle. Crystals – dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Paper 2017/633, 2017. `https://eprint.iacr.org/2017/633`.

[63] S. Even and Y. Mansour. A construction of a cipher from a single pseudorandom permutation. In H. Imai, R. L. Rivest, and T. Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, Nov. 1993.

[64] A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves draft-irtf-cfrg-hash-to-curve, `https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/`, June 2020.

[65] A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves, irft-cfrg active draft, `https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/`, 2022.

[66] P.-A. Fouque, A. Joux, and M. Tibouchi. Injective encodings to elliptic curves. In C. Boyd and L. Simpson, editors, *ACISP 13*, volume 7959 of *LNCS*, pages 203–218. Springer, Heidelberg, July 2013.

[67] B. Freitas, Y. Gu, and S. Jarecki. Randomized half-ideal cipher on groups with applications to uc (a)pake. In C. Hazay and M. Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 128–156, Cham, 2023. Springer Nature Switzerland.

[68] B. Freitas Dos Santos, Y. Gu, S. Jarecki, and H. Krawczyk. Asymmetric pake with low computation and communication. In *EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022.

[69] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, Aug. 1999.

[70] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO'99*, pages 537—554, 1999.

[71] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. Cryptology ePrint Archive, Paper 2003/032, 2003. `https://eprint.iacr.org/2003/032`.

[72] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology – CRYPTO 2006*, pages 142–159. Springer, 2006.

[73] A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 516–525, New York, NY, USA, 2010. Association for Computing Machinery.

[74] Y. Gu, S. Jarecki, and H. Krawczyk. KHAPE: Asymmetric PAKE from key-hiding key exchange. In *Advances in Cryptology - Crypto 2021*, pages 701–730, 2021. `https://ia.cr/2021/873`.

[75] C. Guo and D. Lin. Improved domain extender for the ideal cipher. *Cryptography Commun.*, 7(4):509–533, dec 2015.

[76] B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. In *Conference on Cryptographic Hardware and Embedded Systems 2019 – CHES 2019*. IACR, 2019. Cryptology ePrint Archive, Report 2018/286.

[77] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, 1999.

[78] S. Halevi and H. Krawczyk. One-pass hmqv and asymmetric key-wrapping. In *Advances in Cryptology – PKC 2011*, pages 317–334. Springer, 2011.

[79] F. Hao and S. F. Shahandashti. The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585, 2014. `http://eprint.iacr.org/2014/585`.

[80] J. Hoffstein, J. Pipher, and J. H. Silverman. Ntru: A ring-based public key cryptosystem. In J. P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[81] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. https://eprint.iacr.org/2017/604.

[82] T. Holenstein, R. Künzler, and S. Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC*, pages 89–98. ACM Press, June 2011.

[83] A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, and W. Beullens. Sphincs+ submission to the nist post-quantum project, `https://sphincs.org/data/sphincs+-specification.pdf`, 2017.

[84] J. Y. Hwang, S. Jarecki, T. Kwon, J. Lee, J. S. Shin, and J. Xu. Round-reduced modular construction of asymmetric password-authenticated key exchange. In *Security and Cryptography for Networks – SCN 2018*, pages 485–504. Springer, 2018.

[85] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

[86] D. P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.

[87] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In *Applied Cryptology and Network Security – ACNS 2017*, pages 39–58. Springer, 2017.

[88] S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In *Advances in Cryptology - EUROCRYPT 2018*, pages 456–486, 2018. IACR ePrint version at http://eprint.iacr.org/2018/163.

[89] S. Jarecki, H. Krawczyk, and J. Xu. On the (in)security of the diffie-hellman oblivious PRF with multiplicative blinding. In *Public-Key Cryptography - PKC 2021*, pages 380–409, 2021. `https://ia.cr/2021/273`.

[90] É. Jaulmes, A. Joux, and F. Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In J. Daemen and V. Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 237–251. Springer, Heidelberg, Feb. 2002.

[91] S. Jiang, G. Gong, J. He, K. Nguyen, and H. Wang. Pakes: New framework, new techniques and more efficient lattice-based constructions in the standard model. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 396–427, Cham, 2020. Springer International Publishing.

[92] C. S. Jutla and A. Roy. Smooth NIZK arguments. In *Theory of Cryptography – TCC 2018*, pages 235–262. Springer, 2018.

[93] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology — EUROCRYPT 2001*, pages 475–494, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[94] J. Katz and V. Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices, 2009.

[95] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 252–267. Springer, Heidelberg, Aug. 1996.

[96] T. Kim and M. Tibouchi. Invalid curve attacks in a GLS setting. In K. Tanaka and Y. Suga, editors, *IWSEC 15*, volume 9241 of *LNCS*, pages 41–55. Springer, Heidelberg, Aug. 2015.

[97] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *1996 Internet Society Symposium on Network and Distributed System Security (NDSS)*, pages 114–127, 1996.

[98] H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Advances in Cryptology – CRYPTO 2003*, pages 400–425. Springer, 2003.

[99] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, Aug. 2005.

[100] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol (extended abstract). In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.

[101] H. Krawczyk, K. Lewi, and C. Wood. The opaque asymmetric pake protocol, draft-irtf-cfrg-opaque, `https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/`, Feb. 2021.

[102] V. Lyubashevsky. comparison, `https://pq-crystals.org/dilithium/data/slides-pqcrypto17-lyubashevsky.pdf`, 2017.

[103] V. Lyubashevsky. comparison, `https://csrc.nist.gov/CSRC/media/Presentations/crystals-dilithium-round-3-presentation/images-media/session-1-crystals-dilithium-lyubashevsky.pdf`, 2021.

[104] V. Lyubashevsky, N. K. Nguyen, and M. Plancon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. Cryptology ePrint Archive, Paper 2022/284, 2022. `https://eprint.iacr.org/2022/284`.

[105] P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. `http://eprint.iacr.org/2001/057`.

[106] V. Maram, P. Grubbs, and K. G. Paterson. Anonymous, robust post-quantum public key encryption. In *EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022.

[107] M. Marlinspike. Simplifying OTR deniability, https://signal.org/blog/simplifying-otr-deniability/, 2013.

[108] M. Marlinspike and T. Perrin. The X3DH key agreement protocol, https://signal.org/docs/specifications/x3dh/, 2016.

[109] I. McQuoid, M. Rosulek, and L. Roy. Minimal symmetric PAKE and 1-out-of-n OT from programmable-once public functions. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.*, 2020.

[110] I. McQuoid and J. Xu. An efficient strong asymmetric pake compiler instantiable from group actions. Cryptology ePrint Archive, Paper 2023/1434, 2023. https://eprint.iacr.org/2023/1434.

[111] R. C. Merkle. One way hash functions and DES. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 428–446. Springer, Heidelberg, Aug. 1990.

[112] NIST Information Technology Lab. Post-quantum cryptography, https://csrc.nist.gov/projects/post-quantum-cryptography.

[113] D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In R. Karri, O. Sinanoglu, A.-R. Sadeghi, and X. Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, Apr. 2017.

[114] D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In *ASIACCS 17*, pages 301–312. ACM Press, 2017.

[115] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, Aug. 1994.

[116] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Fast-fourier lattice-based compact signatures over ntru, https://falcon-sign.info/, 2017.

[117] A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, 2006.

[118] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.

[119] V. Shoup. Security analysis of SPAKE2+. *IACR Cryptol. ePrint Arch.*, 2020:313, 2020.

[120] N. Sullivan, H. Krawczyk, O. Friel, and R. Barnes. Opaque with tls 1.3, draft-sullivan-tls-opaque, https://datatracker.ietf.org/doc/draft-sullivan-tls-opaque/, Feb. 2021.

[121] M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 139–156. Springer, Heidelberg, Mar. 2014.

[122] R. S. Wahby and D. Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES*, 2019(4):154–179, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8348`.

[123] R. S. Winternitz. Producing a one-way hash function from DES. In D. Chaum, editor, *CRYPTO'83*, pages 203–207. Plenum Press, New York, USA, 1983.

[124] K. Xagawa. Anonymity of nist pqc round 3 kems. In *EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2022.

[125] J. Zhang and Y. Yu. Two-round pake from approximate sph and instantiations from lattices. In *ASIACRYPT (3)*, pages 37–67. Springer, 2017.