Attacks and security proofs of authenticated key-exchange protocols

Petra Šala

PSL UNIVERSITÉ PARIS

ENS | PSL

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université du Luxembourg
Dans le cadre d'une cotutelle avec, École normale supérieure

Attaques et preuves de sécurité des protocoles d'échange de clés authentifiés

Attacks and Security Proofs of Authenticated Key-Exchange Protocols

Soutenue par

Petra ŠALA
Le 15 Septembre 2021

Ecole doctorale n° 386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Jean-Sébastian, CORON Professor, University of Luxembourg	*Président*
Kristian, GJØSTEEN Professor, Norwegian University of Science and Technology	*Rapporteur*
Marc, JOYE Chief Scientist, Zama	*Rapporteur*
Michel, ABDALLA Professor, École normale supérieure	*Examinateur*
Manuel, BARBOSA Assistant Professor, University of Porto	*Examinateur*
Whitfield, DIFFIE Consulting Professor Emeritus, Stanford	*Examinateur*
David, NACCACHE Professor, École normale supérieure	*Directeur de thèse*
Peter Y.A. RYAN Professor, University of Luxembourg	*Directeur de thèse*

uni.lu
UNIVERSITÉ DU LUXEMBOURG

"We stand today on the brink of the revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high-grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals.
In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels... At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science."

Diffie and Hellman, 1976

Abstract

The vast majority of communication on the Internet and private networks heavily relies on Public-key infrastructure (PKI). One possible solution, to avoid complexities around PKI, is to use Password Authenticated Key-Exchange (PAKE) protocols. PAKE protocols enable a secure communication link between the two parties who only share a low-entropy secret (password). PAKEs were introduced in the 1990s, and with the introduction of the first security models and security proofs in the early 2000s, it was clear that PAKEs have a potential for wide deployment - filling the gap where PKI falls short. PAKEs' PKI-free nature, resistance to phishing attacks and forward secrecy are just some of the properties that make them interesting and important to study. This dissertation includes three works on various aspects of PAKEs: an attack on an existing PAKE proposal, an application of PAKEs in login (for password leak detection) and authentication protocols (HoneyPAKEs), and a security analysis of the J-PAKE protocol, that is used in practice, and its variants.

In our first work, we provide an empirical analysis of the zkPAKE protocol proposed in 2015. Our findings show that zkPAKE is not safe against *offline dictionary* attacks, which is one of the basic security requirements of the PAKE protocols. Further, we demonstrate an implementation of an efficient offline dictionary attack, which emphasizes that, it is necessary to provide a rigorous security proof when proposing a new protocol.

In our second contribution, we propose a combined security mechanism called HoneyPAKE. The HoneyPAKE construction aims to detect the loss of password files and ensures that PAKE intrinsically protects that password. This makes the PAKE part of the HoneyPAKE more *resilient* to *server-compromise* and *pre-computation* attacks which are a serious security threat in a client-server communication. Our third contribution facilitates the wider adoption of PAKEs. In this work, we revisit J-PAKE and simplify it by removing a *non-interactive zero knowledge proof* from the last round of the protocol and derive a lighter and more efficient version called *sJ-PAKE*. Furthermore, we prove sJ-PAKE secure in the indistinguishability game-based model, the so-called *Real-or-Random*, also satisfying the notion of *perfect forward secrecy*.

Acknowledgements

First and foremost, my gratitude goes to Professor Peter Y.A. Ryan, who hired me and entrusted me to do the research work that resulted in this thesis. I thank Prof. Ryan for his inspiration and continuous support throughout my scientific journey. In all challenging moments, I could turn to him for advice and guidance. Furthermore, I am indebted to him for the hard-working and friendly environment in the research group and funding my participation in EUROCRYPT'17, LATINCRYPT'17, CRYPTO'18 and ESORICS'19. Attending those conferences allowed thrilling discussions and efficient exchange of ideas with distinguished cryptographers. The scientific collaborations born at these encounters significantly impacted my vision, research and hence my PhD.

I am profoundly honored to have Turing Award winner Prof. Whitfield Diffie in my defense committee. Prof. Diffie was the first "big name" I came across when I started my studies in cryptography, and when I discovered for the first time his ground-breaking 1976 paper, "*New Directions in Cryptography*" I was far from imagining that one day I would have an opportunity to meet him and have the privilege to discuss my work with him. Nevertheless, the seminal, signed by Prof. Diffie and his co-author Martin Hellman achieved eternal fame. Moreover, it yielded the eponymous Diffie-Hellman key exchange, on which many constructions (including mine) are based.

My thanks also go to my co-supervisor, Professor David Naccache, who initiated my crypto journey and kept inspiring me to learn languages and invent creative crypto ideas. I am very grateful to him for arranging a co-tutorship with École normale supérieure de Paris (ENS Paris) and for believing in my abilities. His witty sense of humor often made me relaxed.

I would like to thank Dr Peter Browne Rønne, for his invaluable guidance, support and encouragement. I appreciate all his help and preparation every time before my travels to Paris.

I am also grateful to Professor David Pointcheval, head of the ENS's Computer Science Department and Cryptography Laboratory, who admitted me to the ÉNS' doctoral selective program and to whom I could always turn to when I had questions.

I also thank the University of Luxembourg and ÉNS Paris for funding my numerous research travels to Paris. Those short stays allowed me to refine my scientific results. I appreciate every moment spent at ÉNS and consider myself lucky to have gotten the best from both institutions.

I express my affection to the numerous members of the APSIA team from the University of Luxembourg and the ÉNS' Cryptography and Security teams. I am grateful for all the time we spent discussing and encouragement before the submission deadlines.

Words can hardly express my gratitude towards my collaborators, Professor Michel Abdalla and Professor Manuel Barbosa for their patience and understanding and for

answering each of my questions.

I want to thank my colleague and a friend Dr Marjan Škrobot for all our discussion and guidance, and for understanding how it is when tackling the problem for the first time.

I am very thankful to Dame Natacha Laniado, for her support and friendship during my PhD studies and for always believing in me.

I am indebted to my best friend Dr Sasan Jafarnejad, who taught me to dig deep to find solutions to problems, who always provided constructive criticism and helped me become the researcher and person I am today. I would also like to thank my closest friends Maja and Vanda, for their unconditional support during every challenge met and every success I made during the past four years.

Most of all, I would like to thank my family, especially my beloved parents, my mother Zlata and my father Stjepan. They constantly reminded me that every problem has a solution and that there is nothing I cannot solve. I would also like to thank my brother Hrvoje, who always supported and admired my work.

This work was conducted at the APSIA group of the University of Luxembourg's Faculty of Science, Technology and Medicine (FSTM) and École normale supérieure, Paris.

Contents

List of Figures

List of Tables

"An expert is a person who has made all the mistakes that can be made in a very narrow field."

Niels Bohr

1 Introduction

1.1 Key Management and the Public-Key Revolution

Until about fifty years ago, cryptography (namely symmetric cryptography) has been used almost exclusively for governmental, diplomatic and military purposes. The primary use of symmetric cryptography (presently also known as private-key cryptography) was to ensure the secrecy and integrity of the messages sent over an insecure channel. To achieve these ends, the parties would need to agree upon a secret key in advance. For centuries (if not millennia), humans struggled with ways to share keys securely. Typically, this would be achieved either by moving one party physically towards the other (meeting) or moving the key physically. In some situations, the parties used courier services to transport long keys. A nice example of such practice is given in [KL07]: the "red phone" that connected Washington and Moscow in the 1960s, was encrypted with keys transported by couriers who moved around the globe, carrying briefcases with long printed keys[1]. However, with the advent of electronic communication, one could not rely anymore on the methods mentioned above, given their impracticality, inconvenience, and logistical clumsiness. To illustrate this, let's consider the case of a company with N employees wishing to converse privately with each other using private key cryptography. Within this group, each employee would need to manage and store $N-1$ secret keys. This places a significant burden on IT administrators and users in terms of

[1]Printed paper keys are still used today, and only discontinued recently in the US https://techmonitor.ai/techonology/cybersecurity/punched-tape-ukkpa

key management: preserving the secrecy of many keys, key updates and key revocations are just some of the issues that need to be addressed (see e.g. [BOM+19]) and [JY98] for alternative approaches). Introducing a Key-Distribution Center (KDC) as a trusted entity that helps manage secret keys for employees only partially fixes the problem. One improvement is that with relying on KDC, employees need to store only *one* long-term secret key (shared with KDC) and the other secret keys are short-term and erased once the communication between the employees ends. One downside is that since all the trust is given to KDC, a successful attack on KDC would result in a complete breakdown of the system. The well-known *Kerberos* [NYHR05] is a good example of a still-used KDC. However, the Kerberos protocol architecture is not suitable for open systems like the Internet as it requires a continuous online presence of KDC. Moreover, in open systems the users typically commence the communication without prior key setup between each other[2]. The first ones who suggested a different approach to the key distribution and management with a ground-breaking work were Whitfield Diffie and Martin Hellman in 1976, "New Directions in Cryptography". Diffie and Hellman changed the course of cryptography by setting foundations for public-key cryptography and made cryptography accessible to any person with Internet access. Furthermore, public-key cryptography yielded new key-exchange protocols that enable establishing a secret key between honest parties over a public channel. The biggest advantage of the public-key crypto is that anyone can encrypt a message with the public key, but only the person in possession of the private key can decrypt and read the message. Later, it was acknowledged that the established secure channel between the parties must also be authenticated to achieve a communication secure even with the adversary's interference. The *Diffie-Hellman Key Exchange protocol (DH-KE)* marked the beginning of standardized key-exchange protocols in a public-key setting. Even though the original DH-KE is not used as it is prone to a *man-in-the-middle attack*, it serves as a core of key-exchange protocols based on public-key which are resilient to man-in-the-middle attacks and are widely used in practice. One remarkable example where those protocols are used is Transport Layer Security (TLS) protocol [DR08]. Finally, we stress the main advantages of public-key cryptography over private-key cryptography:

- Public-key cryptography allows keys distribution to be done over public (but authenticated) channels.
- Public-key cryptography reduces the need to store many secret keys.
- Public-key cryptography is more suitable for parties who have never previously interacted which is a perfect fit for *online* communication, thus enabling e-commerce.

[2]Public-key infrastructure, offline TTP

Now that we clearly stated the main differences between the private-key and the public-key cryptography, we give an insight into where the Password Authenticated Key-Exchange (PAKE) protocols fit in. PAKE protocols use a low-entropy password to derive a strong cryptographic key shared between two parties over an insecure channel. In other words, PAKEs rely on public-key cryptography, while the long-term secret (i.e. password) is symmetric in nature.

1.1.1 Provable Security Paradigm

The profound expansion of Modern Cryptography into a public-key setting also encouraged the development of interactive protocols more systematically. The cryptographic schemes are analyzed with a goal to precisely define their security. This means that every scheme should undergo rigorous analysis showing that a given construction is secure. Security proofs typically rely on *assumptions* that need to be made explicit and mathematically precise. Public-key cryptography yielded new mathematical assumptions and primitives which form the basis for proving a novel scheme secure. A proof of security should be constructed to rely on the hardness of the assumption which is considered to be unbreakable. In other words, the underlying scheme is secure if the adversary cannot break the assumptions on which the scheme in question relies. More importantly, the more adversarial capabilities the proof captures, the more confidence cryptographers and developers will have to use a real-world deployment schemes. Inspired by Diffie-Hellman, many new mathematical assumptions have been proposed and are often used in security proofs of interactive key-exchange protocols. To give an example on an intuitive level, the key-exchange protocol where two parties agree on the same session key is considered secure in the view of an eavesdropping adversary, if the derived keys look completely random when shown to the adversary. This is a crux of an *indistinguishability-based* model on which the majority of the PAKE protocols base their security proofs and was first formalized in the 1990s.

1.2 Design of Key-Exchange Protocols

The fundamental role of key-exchange protocols is to establish a secure channel between two communicating parties via the negotiation of a shared key. Key-exchange protocols are the most common form of a cryptographic protocol in wider use, teaching us a lot about designing and analyzing more complex protocols. When designing a key-exchange protocol, one needs to ensure that parties know exactly whom they are talking to and that the protocol does not leak any information about the resultant shared key. We consider the adversaries who monitor, control or modifies traffic. Furthermore, the adversaries may corrupt the parties (e.g. retrieve the long-term secret) or learn

session-specific information (session keys) for various reasons. In other words, every key-exchange protocol should confine the damage of exposure to a minimum. We present a list of the requirements when designing a proper key-exchange protocol:

- The security model should capture the full capabilities of realistic attackers;
- The protocol should be easy to analyze, both as standalone and in composition with other protocols;
- The protocol should have formal security proof in the *state-of-the-art* model;
- The security model should capture forward secrecy;
- The protocol should have low computation and communication costs;
- The protocol should avoid overkill requirements, and it should have a robust implementation.

1.2.1 Authentication

The possibility of establishing confidential channels between remote entities without having to distribute key material in advance securely was proposed by Diffie and Hellman [DH76]. However, mechanisms such a Diffie-Hellman key establishment still require some mechanism to securely authenticate the parties participating in the protocol and avoid man-in-the-middle attacks. Various ways of incorporating authentication to such protocols have been proposed, ranging from approaches that rely on symmetric or asymmetric long-term setup to those using additional out-of-band channels (e.g., Threema [Thr12]). Asymmetric approaches include the use of digital signatures or incorporating long term key material into the computation of the session key, as in the MTI class of protocols, [MTI86]. However, these assume some way for the parties to be confident of the public keys associated with the other parties, thus requiring either some form of a trusted setup (e.g., PKI or a Web of trust, etc). A further possibility is to have the parties share a (possibly low-entropy) secret in advance and use it to bootstrap the authenticity of subsequently established session keys. As another possibility one could use out-of-band channels, authenticated by some other means such as line of sight or distance bounding, to confirm the identity of parties involved in the protocol. In this work we consider a low-entropy, symmetric, long-term setup, which is the most widely used setting for human authentication.

1.3 Password-based Authentication

The way how passwords are used for authentication purposes has been evolving for more than sixty years. In the beginning, passwords were just used for system entries

and prevented users from accessing resources that exceeded their authorization. Later, in the mid-1990s, the attention was on replacing passwords with client certificates *via* newly proposed SSL. However, this approach failed due to bad issues with the management of clients certificates and private keys. Moreover, applying passwords on web-based services triggered new obstacles such as one-sided authentication and phishing attacks, among many. The desire and need for stronger security measures encouraged some companies to develop additional authentication tools such as hardware tokens or smartphones in the 2010s. Still, many studies on models deal with choosing a strong password [BHvOS12, Bon12], measuring a desirable password entropy, studying user's behavior, shielding weak passwords and much more. Despite many well-known limitations of password-based authentication [GEAR09] this is still the most prevalent way of user authentication online. Some of the existing password-based authentication methods [Skr17]:

1. **Passwords in plain text**. Until the early 2000s, it was common practice to use authentication mechanisms by which passwords are sent in plain text over an insecure network. The Basic Access authentication method used for client authentication when making a connection request in Hypertext Transfer Protocol (HTTP) is probably the best-known example.

2. **Hash of password.** Another approach, which by today's standards is insecure but persists in some network protocols, consists of sending a hash value of a user's credentials (username and password) over an insecure network. An old mechanism that follows this strategy to solve the password authentication problem is Challenge Handshake Authentication Protocol (CHAP).

3. **A Password over TLS.** Currently, the Web's most prominent password authentication mechanism on the Web is what Manulis et al. [MSKD16] call HTML-forms-over-TLS. It works as follows: first, a Transport Layer Security (TLS) channel is established, usually between a client (e.g., user's browser or mobile application) and a service providing server. Then, the server sends an HTML form over the already established TLS channel (encrypted and server-authenticated) to the client, who is then asked to input its password and username to authenticate. Although considered secure, this mechanism can be affected by PKI-related security issues or phishing attacks, for example. These weaknesses can help attackers circumvent the existing security mechanisms and get into the possession of users' credentials.

4. **Using Password-Authenticated Key Exchange (PAKE).** PAKE is a cryptographic primitive that offers a way to bootstrap a low-entropy password into a high-entropy key without relying on PKI. It is a mature method for password authentication that can be in some scenarios used on its own [AP05, HR10, Mac02]

or as a building block of more complex protocols (e.g., TLS and PAKE integration [MSKD16]). However, even though it was proposed more than 25 years ago, PAKEs have been only recently considered for wide-scale use.

5. **Two-factor authentication** A desire for stronger security measures encouraged some services providers to include, in addition to passwords, other authentication tools such as hardware tokens or smart-phones [SJ15]. Another authentication method worth mentioning is a device (e.g. YubiKey[3]) which supports FIDO2 [BBCW20] protocols and the only way the user authenticates himself, in conjunction with a password, is with *a simple touch* on the device. However, these new approaches usually leverage the time component and use the additional secure channel as well, making the task for attackers much harder.

1.3.1 Password Authenticated Key-Exchange Protocols under Patents

Password authenticated key exchange protocols are deployed as widely in practice as could have been hoped for (e.g. e-passports use CPACE [BK09]). While some think that the reason is that they are not well known, others think that it is related to patent issues. For example EKE and PAK were patented by Lucent Technologies [12], SPEKE by Phoenix Technologies [18] and SRP by Stanford University [28]. As a result new PAKE protocols were proposed, among which J-PAKE [HR10] and OPAQUE [JKX18] are worth mentioning. We believe that now that all patents are expired, there should be increased motivation for a broader use of PAKEs, especially for internet-applications where password-based authentication mechanism is common (e-banking and e-commerce).

1.3.2 Security Standardization of Password Authenticated Key-Exchange Protocols

The most relevant standards used for the standardization of password authenticated key-exchange protocols are ISO [ISO09], IEEE [IEE02] and IETF [IET]. While SPEKE [Jab96] is specified in the ISO/IEC 11770-4 and IEEE P1363.2, PAK and its variants [Mac02] have been included in IEEE standards; some other PAKE protocols were recently under consideration for standardization by the IETF [HR10, JKX18, AHH21]. In addition, the Crypto Forum Research Group (CFRG), an IRTF (Internet Research Task Force) research group, recently conducted a competition to select the password authenticated key-exchange protocol for standardization. The selection process gathered experts from industry and academy who could nominate any PAKE protocol and participate in discussions and reviews of proposed candidates. After ten months, two candidates of password authenticated key exchange were chosen as winners: CPace ([HL18], [AHH21])

[3] [Yub]

and OPAQUE [JKX18]. We lay out more details on IETF standardization in Chapter 3.

1.4 Motivation for using PAKEs

Due to being free from public-key infrastructure (PKI), PAKEs protocols have a good use in practice. Below we provide three different scenarios in which PAKEs are put to use: in the Internet of Things (IoT) as a part of Device Provisioning Protocols (DPP), within a login mechanism, and as a part of the software for file transfer. We supply a concrete example for each scenario and point out exactly which PAKE protocol is used.

1.4.1 Device Provisioning

PAKE protocols are increasingly used is in the IoT networks, where authentication and device synchronization should be as simple as possible. Most devices in IoT networks for authentication use low-entropy secrets to avoid complexities around PKI, which is usually a four-digit PIN. Thus, a perfect fit for PAKEs. A concrete example that uses the elliptic curve version of J-PAKE is Thread Protocol [Thr16]. J-PAKE protects the communication from malicious actors by preventing offline dictionary attacks which is the basic requirement that PAKE protocols need to fulfil. Furthermore, J-PAKE ensures the secure channel where the PIN is well protected and yields a strong cryptographic key which used for encrypt further communication between the devices.
Another example that motivates further use of PAKEs is Wi-Fi Easy Connect [Wif18], developed by Wi-Fi Alliance in 2018. Wi-Fi Easy Connect offers devices a simpler way to connect them to a Wi-Fi Network while still maintaining the high-security protection. We stress that among these devices there are also the ones without an interface (e.g., printers) that usually have a more complicated procedure for connecting them to a wireless network than a typical smartphone. Wi-Fi Easy Connect uses PKEX [Har18] (based on SPAKE2 protocol [AP05]) as a password authentication mechanism to exchange public keys, which is included in DPP.[4]

1.4.2 Login Scenarios

Another potential use of PAKE protocols is in login systems. Usually, a client wants to authenticate himself to a server, to access a particular website provided by the server. The client is the one who decides to trust the server and inputs his credentials like username and password directly to the server's website. We assume that the server

[4]DPP is a protocol developed by Wi-Fi Alliance that implements Wi-Fi Easy Connect. The exact specification draft can be found in [Wif18].

stores passwords[5]. Here, we explain potential problematic scenarios [Bre19] and we discuss a solution offered by PAKE below:

- Failure of PKI. A Certificate Authority (CA) could issue void certificates, compromising the user's passwords. An example of this issue occurred in 2011, when a dutch CA company DigiNotar [Den11], was compromised, causing impostor websites to have valid certificates. Afterwards, domains such as Google and Mozilla removed DigiNotar root certificates from their list of trusted roots.

- Phishing attacks. The malicious attacker can obtain a user's credentials through a fraudulent website that has a valid certificate.

We stress that the *client-server* setting becomes problematic when the connection with an adversary impersonating the server is marked as "secure" due to a valid certificate that the adversary can obtain through a CA authority. Moreover, we highlight that TLS just ensures the secure transit of a password but does not prevent an attacker with a valid certificate to impersonate the server. Fortunately, PAKE protocols do not rely on PKI and they prevent phishing attacks by authenticating a client to a server *without* exposing the password. The following cases show why PAKE protocols are a good fit for login scenarios [OWT09, EKSS09]. First, we propose establishing a TLS-connection between the client and a server and then running a PAKE protocol as part of the TLS-channel, allowing the client to authenticate himself without sending a password. Then, if the adversary who holds a valid certificate is impersonating the server, the connection will fail when the password between the fake website and a client would not match.
Then, we suggest another login scenario in which PAKE is integrated into TLS, i.e., for the client's web authentication. The main aim of deploying PAKE for web authentication would be to create a safe and trusted path for the users to input their credentials and thus mitigate phishing attacks. In addition, authentication through PAKE in a *client-server* setting would prevent disclosing the client's password even when the attack occurs. However, it is still uncertain whether to deploy PAKE into the application layer (HTTPS-PAKE [OWT09]) or transport layer (SRP-PAKE [TWMP07]). In [EKSS09], the authors addressed some of the usability and deployment issues on integrating PAKEs into TLS. Nevertheless, PAKE-based web authentication shows interesting properties and reasonable solutions against social engineering attacks. The proof that PAKEs successfully pave a path for its widespread adoption in TLS is the works [PS20] and [SKFB21], where SPAKE2 and OPAQUE are proposed as a part of TLS 1.3.

[5]To avoid immediate password exposure, passwords are usually stored on the server as a hash of a password or hash of a password together with a unique salt related to only one per password. Later, we provide more details in Chapter 3.

1.4.3 File Transfer System

PAKE has shown to be an efficient and practical solution for several different scenarios. A final example that supports the usage of PAKEs, but differs from the previous examples is when two parties want to exchange a file between themselves over an insecure network using a command line. We assume that both parties share a pre-agreed (possibly one-time) password or even a PIN and are online simultaneously. An example of software that can ensure a safe file transfer while conditions above are fulfilled is a Magic-Wormhole [Mag16]. To achieve an authentication of the users involved in the file transfer and offline dictionary attacks resistance during the initial connection attempt, it uses the SPAKE2 protocol [AP05]. Magic-Wormhole is a straightforward way to transfer files without any login or web interface despite the various available software. For these reasons, we hope our work will encourage and motivate its wider use.

1.5 Research Goals

We highlighted some potential risks when passwords are used as an authentication mechanism and hinted at the solutions when using PAKEs. In our first work, *An offline dictionary attack against zkPAKE protocol* in Chapter 5, we give an example of the proposed PAKE protocol without security proof and provide a concrete and practical *offline dictionary attack*. This work was accepted and presented on IFIP[6], 2019.
To support the idea of fitting in PAKE to a login scenario, *HoneyPAKEs* in Chapter 6, provide a login mechanism, HoneyPAKE, which involves the following: a PAKE protocol, a simple access control mechanism, and a set of decoy passwords called *Honeywords*. Furthermore, we show that HoneyPAKE is resistant to server-compromise and pre-computation attacks, which is why the HoneyPAKE protocol has a real potential to be considered for real-world deployment. This work was accepted and presented on *Security Protocols XXVI - 26th International Workshop* in 2018.
In our third work, *Security characterization of J-PAKE and its variants* in Chapter 7, we revisited one of the most widely used PAKE protocols, J-PAKE. In addition, we created a simplified version, *sJ-PAKE*, by removing one zero-knowledge proof from the original J-PAKE (on both sides) and provided a security proof of sJ-PAKE in the *indistinguishability-based model*. By providing a security proof of sJ-PAKE and confirming that sJ-PAKE indeed fulfills the standard requirements, we comply with the latest security guidelines for standardization which will encourage sJ-PAKE for deployment in practice. In addition, we provide security proofs of sJ-PAKE's variants and discuss their differences. In the end, we compare the efficiency of the original J-PAKE to sJ-PAKE

[6] 34th IFIP TC-11 SEC 2019 International Conference on Information Security and Privacy Protection

and conclude that the sJ-PAKE version is more efficient, which puts it higher on the ladder of PAKE protocols waiting to be deployed. This work is currently under submission, but we refer to the eprint version [ABR+21]. For our future work, we explore benefits of adding explicit authentication to sJ-PAKE protocol by using confirmation codes. Our initial analysis shows that we can achieve simpler proof compared to sJ-PAKE without confirmation codes.

1.6 Outline

The outline of the dissertation is the following:

Chapter 2: Preliminaries.

This chapter covers all cryptographic building blocks needed to understand better the work we present in later chapters.

Chapter 3: Password authenticated key-exchange protocols.

In this chapter we explain in detail Password Authenticated Key-Exchange (PAKE) protocols, their application, and formally define all security requirements a PAKE needs to fulfil to be considered for deployment.

Chapter 4: Security models for password authenticated key-exchange protocols.

This chapter analyses complexity-theoretic security models of PAKE protocols and covers the Indistinguishability-based Real or Random Model.

Chapter 5: An offline dictionary attack against the zkPAKE Protocol.

This chapter demonstrates, both theoretically and practically, an offline dictionary attack against the zkPAKE protocol.

Chapter 6: HoneyPAKEs.

This chapter lays out a new cryptographic mechanism that relies on PAKE protocols and an additional device that detects whenever the password might be compromised.

Chapter 7: Security characterization of J-PAKE and its variants.

This chapter presents a new variant of the J-PAKE protocol called sJ-PAKE and its security proof in indistinguishability-based Real-or -Random Model. Furthermore, we yield new variants of sJ-PAKE, sRO-PAKE and sCRS-J-PAKE, and prove them in the same fashion as sJ-PAKE. In addition, we layout the game-based code for games and reductions in Supplementary material.

Chapter 8: Conclusion and future work.

We conclude this dissertation with another idea of the protocol, sJ-PAKE protocol with confirmation codes. Then, we compare sJ-PAKE without confirmation codes and sJ-PAKE with confirmation codes, and we give a sketch of the proof of the latter.

If you can't explain it simply, then you don't understand it well enough.

Albert Einstein

2 Preliminaries

2.1 Introduction

This chapter provides background concepts and definitions to make this dissertation self-contained and easier to follow. Furthermore, we describe mathematical definitions, cryptographic primitives, and hardness assumptions required to describe protocols and in-depth analyse the security proof we provide in Chapter 7.

2.2 Mathematical Background

Notation. We use calligraphic letters to denote adversaries, typically $\mathcal{A}$. We write $d \xleftarrow{\$} D$ for sampling uniformly at random from set D, and $|D|$ to denote the number of elements in D. Let $\{0,1\}^*$ denote the bit string of arbitrary length while $\{0,1\}^\ell$ stands for those of length ℓ. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in [0 ... q-1], and all operations on these are performed mod q.

2.2.1 Groups

Here, we define groups and their properties [KL07].

Definition 2.1. Let $\mathbb{G}$ be a set where $\circ$ is a binary operation between two elements of $\mathbb{G}$. Then, $(\mathbb{G}, \circ)$ represents a **group** if the following four axioms are fulfilled:

Closure: $\forall\ g, h \in \mathbb{G}$, $g \circ h$ is an element in $\mathbb{G}$.

Existence of a neutral element: $\exists$ a neutral element $e \in \mathbb{G}$ such that for all $g \in \mathbb{G}$ holds that $e \circ g = g = g \circ e$.

Existence of inverses: $\forall g \in \mathbb{G}$, there exists an inverse element $h \in \mathbb{G}$ such $g \circ h = e = h \circ g$, denoted as g^{-1}.

Associativity: $\forall g, h, k \in \mathbb{G}$ it holds that $(g \circ h) \circ k = g \circ (h \circ k)$.

Definition 2.2. (Cyclic Group). Let $\mathbb{G}$ be a group of finite order z. The group $\mathbb{G}$ is *cyclic* if there exists a generator $g \in \mathbb{G}$ such that

$$\{g^0, g^1, ..., g^{z-1}\} = \mathbb{G}. \tag{2.1}$$

We underline that every element $h \in \mathbb{G}$, where $\mathbb{G}$ is a cyclic group with the generator g, can be written as g^n, for some $n \in \mathbb{N}$.

Theorem 2.3. *Let $\mathbb{G}$ be a finite group with $z = |\mathbb{G}|$. Then, $\forall g \in \mathbb{G}$, it holds that $g^z = 1$.*

We stress that finding a discrete logarithm n of $h = g^n \in \mathbb{G}$ can be hard and is considered to be a general problem (*discrete logarithm problem* (DLP)) in modern cryptography.

Prime-order groups. In cryptography, the most interesting groups are those where relevant problems, such as *discrete logarithm* (DLP) (Definition 2.13) and *Diffie-Hellman* problems (DHP), are hard to solve. Prime-order cyclic groups are one example of groups where the DL problem is considered to be hard. One reason is that if a cyclic group did not have a prime order, DL would become easier to solve using the *Pohlig-Hellman algorithm*. Another reason for using a cyclic group is that every group element in prime-order groups is a generator, and any non-zero exponent is invertible. The latter is especially important because some cryptographic constructions require computing multiplicative inverses of certain exponents. Lastly, we stress that the *decisional* Diffie-Hellman problem (DDH) (Definition 2.17) is believed to be hard in prime-order groups. More precisely, given $g^x, g^y, g^z \in \mathbb{G}$, where $\mathbb{G}$ is prime-order group with a generator g and g^x, g^y, g^z are a uniform tuple, it is hard to distinguish whether the element g^z is indistinguishable from a uniform group element.

Subgroups of $\mathbb{Z}_p^*$.[1] As already stressed, working in prime-order groups is advantageous if we want to achieve that problems such as DDH are hard to break. Groups like $\mathbb{Z}_p^*$

[1] An upper script $(*)$ denotes a multiplicative group.

where p is prime, does not have a prime order. More precisely, DDH problem is not guaranteed to be hard in such groups. For his reason, we need the right group structure, such as the *subgroup* of $\mathbb{Z}_p^*$ (defined in Theorem 2.4), which will guarantee the hardness of the DDH problem.

Theorem 2.4. *Let $\mathbb{Z}_p^*$ be a group of order p, such that $p = rq + 1$, with p, q primes. Then*

$$\mathbb{G} := \{[h^r \bmod p] \quad | \quad h \in \mathbb{Z}_p^*\} \tag{2.2}$$

is a subgroup of $\mathbb{Z}_p^*$ of order q.

We will use a subgroup $\mathbb{G}$ of order q with a generator $g \in \mathbb{G}$ in Chapter 7.

Definition 2.5. Let a function $H : \{0,1\}^* \to \{0,1\}^\ell$ be a deterministic cryptographic function. H is called a **(cryptographic) hash function** if the following conditions hold:

- **collision resistance:** It is computationally infeasible to find $m \neq m'$ such that $H(m) = H(m')$.
- **preimage resistance:** Given $y = H(m)$, it is computationally infeasible to find m.
- **second-preimage resistance:** Given m, it is computationally infeasible to find m' such that $m \neq m'$ such that $H(m) = H(m')$.

Lemma 2.6. *(Birthday Problem) Fix $N \geq 0$ and let $x_1, ..., x_q$ be chosen uniformly and independently at random from a set of size N. Then the probability that there exists $i \neq j$ such that $x_i = x_j$, is upper bounded by $\frac{q^2}{2N}$. That is,*

$$\textit{coll(q,N)} \leq \frac{q^2}{2N} \tag{2.3}$$

The birthday problem is a standard statistical term used in many cryptographic constructions and the game-based provable security for authenticated key-exchange protocols (AKE). Let us consider a scenario in which an adversary tries to break a protocol P and sees the same hash values in two different sessions. Here, we use a birthday problem bound for hash functions, meaning that we rely on hardness to find a collision. We bound the adversary's probability in finding collisions between *hash outputs* he receives with the birthday bound and we use in our security proof in Chapter 7, Section 7.4.

2.3 Complexity Theoretic Approaches

To deploy cryptographic protocols for real-world purposes, one needs to undertake a rigorous analysis considering the time and the power of the adversary who might be trying

to learn any information when message material gets leaked. The ideal *information-theoretic* secrecy captures the adversaries with *unlimited* computational power that cannot learn absolutely anything about the message once the ciphertext gets leaked. However, information-theoretic complexity is unnecessarily strong and very hard to achieve. For real-world deployment, protocols generally do not need to guarantee that their security holds for an *unlimited* amount of time. In contrast, the protocol's security needs to hold for a *reasonable* amount of time i.e., *polynomial* time. The notions of efficient adversaries, security parameters, and negligible chances of success are captured by *computational security*. Computational security takes into consideration efficient (running in polynomial time) adversaries and considers protocols to be secure even if there is a *very small chance* for them to fail. More accurately, the very small chance should be so small that, we can disregard it when considering real adversarial success to break the protocol. More importantly, we can decrease this probability efficiently by adjusting the security parameter of the protocol. Two general approaches precisely define the relaxations on the adversary's running time and his chances to potentially succeed in breaking the protocol's security: the *concrete approach* and the *asymptotic approach*.

2.3.1 Concrete Approach

This approach considers the exact amount of time t and a specific amount of computational power needed for the adversary to defeat the particular scheme. As mentioned above, the probability of adversarial success in breaking the scheme's security ϵ should be very small. We stress that (t, ϵ) are concrete numbers, and we emphasize that it is important to precisely define what it means to "break" the security. Providing precise concrete security claims is something computational security leans towards. However, concrete and precise security definitions are hard to achieve, so most schemes rely on the asymptotic approach.

2.3.2 Asymptotic Approach

A security definition in an asymptotic approach requires a definition of what it means to break the security of the protocol and specification of the power of the adversary. Both notions include *security parameters* and *efficient* adversaries [KL07].

The security parameter is an integer that parameterizes, both, cryptographic schemes and involved parties. More precisely, when honest parties generate keys, they take n for the security[2]. For the simplicity of this analysis, one can think of n being the length of the key. Furthermore, the security parameter is known to all parties, including the

[2]Normally, n is chosen by the designer of the system.

adversary. Therefore, to precisely define the security of a particular scheme, we look at the running time of the adversary and its success probability as functions of the security parameter, which contrasts with the concrete approach where we use concrete numbers. Principally, the security parameter, is what differs asymptotic from the concrete approach.

Efficient adversaries. We consider efficient adversaries[3] to be probabilistic[4] algorithms running in time *polynomial* in n. More precisely, there is a polynomial p such that the adversary ends his computing in $p(|x|)$ number of steps, for every input $x \in \{0,1\}^n$. Furthermore, we denote PPT as *probabilistic polynomial time* and in this dissertation, we consider all the adversaries (denoted as A in Chapter 6, and $\mathcal{A}$ in Chapter 7) to be PPT adversaries.

Success Probability. When analyzing the scheme's security, one needs to determine the probability that the efficient adversary defeats the underlying scheme. In the asymptotic approach, the cryptographic scheme is considered to be secure if the adversary has a probability of breaking the system which can be made efficiently small by increasing the security parameter, i.e., the probability is *negligible* (defined in Definition 2.7).

Definition 2.7. (Negligible success probability) A function $f : \mathbb{N} \to \mathbb{R}^+$ is **negligible** if $\forall\ p > 0\ \exists N \in \mathbb{Z}$ such that $\forall\ n > N$, it holds that $f(n) < \frac{1}{p(n)}$.

A negligible function is asymptotically smaller than any inverse polynomial function. Thus, in provable security, the advantage of the adversary must be shown to be negligible to obtain meaningful security proof of the key-exchange underhand. One of the advantages of working with negligible functions is that the result stays negligible when the negligible function is repeated a polynomial number of times. This particular property is defined in the form of a Proposition (Proposition 3.6) in [KL07]. A good example of this property is when we bound the adversary's success in the game-based proof of sJ-PAKE (Chapter 7). The adversaries' capabilities are defined through *queries*, where the number of asked queries can be very high (e.g., 2^{60}). In this case, it is enough to prove that the adversary breaks the scheme with negligible probability. Thus, the number of queries repeated a negligible amount of times is still negligible, implying that the security of the underlying scheme holds. Finally, we define the security of the scheme in the asymptotic approach:

A scheme is secure if for every PPT adversary, attacking the scheme, the probability that the attacker succeeds in the attack is negligible (as defined in Definition 2.7).

[3] We assume that all the honest parties run are efficient.

[4] Probabilistic algorithm means that an algorithm can access an unbiased random bit at each step.

2.4 Model Assumptions

As already mentioned, cryptographic security assumptions play a key role when proving the security of the cryptographic protocols. However, often one needs to adapt the assumption on the environment of the protocol execution to prove the security notions of the underlying construction. Here, we define more formally environments and trust setups, commonly used in the security proofs of key-exchange protocols and other cryptographic primitives. One of the most commonly used idealized assumptions is *Random Oracle* (RO) which treats hash functions as idealized objects [BR93] and always provides a fresh output when a different value is fed into the random oracle. Other primitives, such as block ciphers and one way-permutations, can also be idealized by the random oracle. In some cases, protocols need a trusted setup environment, where a trusted third party generates a random element according to some pre-defined distribution, sometimes called *a public coin*. We call such a setup *Common Reference String* (CRS). In CRS, all participants, including the adversary, have access to, and no extra information about the public coin is available to anyone. Another environment where the adversary is computationally restricted is the *Generic Group Model* (GGM). GGM is often used to capture the algorithms that only use group operations and do not use any knowledge about the group representation. In contrast, *The Algebraic Group Model* (AGM) assumes that the representation of group elements is known and captures algorithms only doing algebraic operations by demanding the algorithm to output a group element. Lastly, we emphasize that RO, AGM and GGM models can be combined with the CRS setup, and one example of the security proof where the RO model is combined with CRS is SPAKE2 [AP05].

2.4.1 Random Oracle Model

Here, we describe a RO model which idealizes hash functions (Definition 2.5). There is a lot of protocols that base their security on the collision-resistant property of hash functions. However, to provide a fully rigorous proof of security, one must assume the protocol is being executed in an environment that always provides a consistent output and makes sure that outputs are random. More precisely, a random oracle treats hash functions as truly random functions. Although, the random oracle model is considered to reflect unrealistic properties, it provides a formal methodology that can be used to analyze proofs in-depth. Moreover, it gives confidence that the scheme in question is somewhat secure. In addition, the security proofs from the random oracle model are seen more as evidence that the underlying scheme does not have true design flaws. In the literature, the random oracle is considered to be a "black box" that responds only when "queried" an $H(x)$ of a given input x (described in Figure 2.1).

For each fresh RO query $H(\cdot)$, the simulator chooses a random string $d \xleftarrow{\$} \{0,1\}^\kappa$ and returns d to $\mathcal{A}$. The simulator administrates all the *query - response* by adding them to the list. If $\mathcal{A}$ already asked for H, the simulator simply retrieves the response d from the list and gives it to $\mathcal{A}$. The number of random oracle queries to H we denote as n_h.

FIGURE 2.1: Simulation of $H(\cdot)$ in the Random Oracle Model

2.4.2 Algebraic Group Model

The AGM model was formally introduced by Pallier et al. in [PV05] and first used constructively to prove the security of J-PAKE protocol [HR10] in [ABM15]. Also, the model was further formalized as the *Algebraic Group Model* (AGM) in [FKL18] and [MTT18]. In essence, given a prime order group $\mathbb{G}$, we say that an adversary is *algebraic* if whenever it outputs a group element, $Z \in \mathbb{G}$, it also outputs a discrete log representation, $r_1, \ldots, r_k$, in terms of all the group elements, $g_1, \ldots, g_k$ that it received so far, i.e. $Z = g_1^{r_1} \cdots g_k^{r_k}$. More formally:

Definition 2.8. Let $\mathbb{G}$ be a cyclic group with the generator g of prime order q. An adversary $\mathcal{A}_{alg}$ is **algebraic** if it outputs a group element $Z \in \mathbb{G}$ such that $Z = \prod_i g_i^{r_i}$, where $g = (g_1, ..., g_n)$ is a list of all group elements that $\mathcal{A}_{alg}$ obtained during the execution so far.

Unlike the GGM, the AGM model efficiently exploits the algebraic structure of the group and, besides equality, also gives the adversary access to an extractor algorithm that outputs the representation of the group element. Thus, AGM is considered to be a weaker model than the GGM. In addition, in AGM several important computational assumptions are proven equivalent, which is not the case in the GGM. For more details on assumptions and comparison AGM to GGM, we refer to [FKL18]. So far, two works, [ABB+20] and [FKL18], are known to use algebraic adversaries in their schemes and we provide the third work with the heavier use of algebraic adversaries, in Chapter 7.

2.5 Zero-Knowledge Proofs

Zero-knowledge proofs are important cryptographic constructions that were formally introduced by Goldwasser and Micali [GMR89]. Intuitively, the zero-knowledge proof protocol consists of a prover and a verifier, where the prover wants to prove to the verifier that some statement is true. Zero-knowledge makes sure that the verifier does not learn anything besides the truthfulness of the statement.

Non-Interactive Zero-Knowledge (NIZK) (extended to the case of labeled Non-Interactive proof system in [ABM15]) is a system where a prover convinces a verifie,r that a statement belongs to a certain class of *languages* in *a single* message. Furthermore, Blum et al. [BFM88] show we can use a common reference string (CRS) crs, known to both the prover and the verifier without changing the definition of NIZK. Let $\mathcal{R}$ be the relation between the statement x and the witness w, which can be checked in PPT, where $\mathcal{R}(x, w)$ outputs 0 or 1. Then, we define $\mathcal{L}$ as an NP language with a witness relation $\mathcal{R}$ such that $\mathcal{L} = \{x \mid \exists w, \mathcal{R}(x, w) = 1\}$.
A tuple $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver})$ is a *NIZK proof system* for $\mathcal{R}$, where a prover convinces a verifier that $x \in \mathcal{L}$, without revealing the witness w. Since the prover does not actively interact with the verifier, it only outputs $\pi \leftarrow \mathsf{Prv}(x, w, \ell)$ (for some label ℓ) to the verifier, who can check if the proof is valid by running Ver which takes π, x and ℓ as input. Furthermore, it outputs 1 if the proof is valid and 0 otherwise. We extend the NIZK by having the relation $\mathcal{R}$ depend on a public, randomly chosen string called *common reference string*, crs. Furthermore, two following conditions should hold:

1. Perfect completeness. This property ensures that every honest prover (for any label ℓ) who knows w, which is in relation to x, can convince the verifier, with the probability equal to 1, that $x \in \mathcal{L}$.

$$\Pr\left[\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa); (\ell, w, \pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : \mathsf{Ver}(\mathsf{crs}, x, \mathsf{Prv}(\mathsf{crs}, x, w, l), l) = 1 \wedge \mathcal{R}(x, w) = 1\right] = 1 \tag{2.4}$$

2. Soundness. This property states that a PPT adversary can convince the verifier that a false statement is true, i.e., $x \in \mathcal{L}$ without knowing w, only with negligible probability ϵ. We define the advantage of the adversary running against the NIZK to be:

$$\mathrm{Adv}^{\mathrm{sound}}_{\mathsf{NIZK}}() := \Pr\left[\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa); (l, x, \pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : \mathsf{Ver}(\mathsf{crs}, x, \pi, l) = 1 \wedge x \notin \mathcal{L}\right] \leq \epsilon \tag{2.5}$$

We define simulation-soundness and extractability properties in the following section.

2.5.1 Simulation-sound Extractable NIZK

In this section, we define simulation-sound extractable NIZK and its properties, according to Groth [Gro06].

Definition 2.9. (labelled) Simulation-sound extractable NIZK ($\mathsf{SE\text{-}NIZK}$) is a system defined by the tuple $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver}, \mathsf{Sim}_1, \mathsf{Sim}, \mathsf{Extract})$ such that $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver})$ is a non-interactive proof system with:

- $\mathsf{Sim}_1(1^\kappa)$ generates common reference string crs and two trapdoors td_s and td_e

- $\mathsf{Sim}(\mathsf{crs}, \mathsf{td}_s, X, l)$ takes crs, td_s, X, some label l, and outputs a simulated proof of knowledge for X, π
- $\mathsf{Extract}(\mathsf{crs}, \mathsf{td}_e, X, \pi, l)$ extracts a witness x for X from a valid proof of knowledge π, considering some label l, trapdoor td_e and common reference string crs, if possible. Otherwise it aborts.

Furthermore, the following properties hold :

1. Knowledge extraction. Let $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver})$ be a NIZK proof system for $\mathcal{R}$. The NIZK system is knowledge extractable if there exists a knowledge extractor $\mathsf{Extract} = (\mathsf{Extract}_1, \mathsf{Extract}_2)$ such that for all adversaries $\mathcal{A}$ we have

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^k) : \mathcal{A}(\mathsf{crs}) = 1\right] = \Pr\left[(\mathsf{crs}, \mathsf{td}_e) \leftarrow \mathsf{Extract}_1(1^k) : \mathcal{A}(\mathsf{crs}) = 1\right] \quad (2.6)$$

and

$$\Pr\Big[(\mathsf{crs}, \mathsf{td}_e) \leftarrow \mathsf{Extract}_1(1^k); (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}); w \leftarrow \mathsf{Extract}_2(\mathsf{crs}, \mathsf{td}_e, x, \pi) : \\ \mathsf{Ver}(\mathsf{crs}, x, \pi) = 0 \vee (x, w) \in \mathcal{R}\Big] = 1$$

This property is usually part of a proof technique called *rewinding* that demonstrates the security of zero-knowledge proofs. More precisely, the simulator can *rewind* the verifier several times using the same randomness to extract the witness. However, we do not consider the rewinding to be possible here as it would contradict unbounded zero-knowledge property, defined below.

2. Unbounded zero knowledge. For all PPT adversaries, there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ and zero-knowledge holds if the adversary cannot distinguish between real (produced by $\mathsf{NIZK.Prv}$) and simulated (produced by Sim) proofs. we define $\mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}()$ as:

$$\Pr\left[\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa) : \mathcal{A}^{\mathsf{Prv}(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1\right] \\ - \Pr\left[\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa) : \mathcal{A}^{\mathsf{Sim}(\mathsf{crs},\mathsf{td}_s,\cdot,\cdot)}(\mathsf{crs}) = 1\right]$$

where $\mathsf{Sim}(\mathsf{crs}, \mathsf{td}_s, X, x, l) = \mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}_s, X, l)$. The oracles $\mathsf{NIZK.Prv}$ and $\mathsf{NIZK.Sim}$ abort if $(x, w) \notin \mathcal{R}$.

3. Simulation-soundness. This property makes sure that even when the adversary sees simulated proofs of possibly false statements, he cannot prove any false statement. Thus, NIZK soundness is preserved.

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Sim}_1(1^k); (x,\pi) \leftarrow \mathcal{A}^{\mathsf{Sim}_2(\mathsf{crs},\tau,\cdot)}(\mathsf{crs}); (x,\pi) \notin Q \wedge x \notin L \wedge \mathsf{Ver}(\mathsf{crs},x,\pi) = 1\right] \approx 0 \tag{2.7}$$

where Q consists of simulation queries and responses (x_i, π_i).

4. Simulation-sound extractability states that we can extract a witness w for x from a valid proof of knowledge π, even after $\mathcal{A}$ sees many simulated proofs. This property combines the *knowledge extraction* and *simulation-sound* property. We define $\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}()$ as:

$$\begin{aligned}\Pr\Big[&(\mathsf{crs},\mathsf{td}_s,\mathsf{td}_e) \xleftarrow{\$} \mathsf{SimExtract}_1(1^\kappa);\\ &(x,\pi) \leftarrow \mathcal{A}^{\mathsf{Sim}_2(\mathsf{crs},\mathsf{td}_s,\cdot,\cdot),\mathsf{Extract}_2^{(g)(\mathsf{crs},\mathsf{td}_e,\cdot,\cdot,\cdot)}}(G,g,\mathsf{crs},\mathsf{td}_e) : \mathsf{Ver}(\mathsf{crs},x,\pi,l) = 1,\\ &\qquad((X,l),\pi) \notin Q \wedge \mathcal{R}(x,\mathsf{Extract}_2(\mathsf{crs},\mathsf{td}_e,x,\pi,l)) = 0\Big]\end{aligned}$$

where Q is a set of query-response pairs $((X,l),\pi)$ for $\mathsf{Sim}_2(\mathsf{crs},\mathsf{td}_s,\cdot,\cdot)$ and $\mathsf{SimExtract}_1$ is an algorithm that outputs $(\mathsf{crs},\mathsf{td}_s,\mathsf{td}_e)$ such that $(\mathsf{crs},\mathsf{td}_s)$ is distributed in the same way as the output of Sim_1.

2.5.2 Schnorr Signatures

A widely used identification scheme based on the hardness of discrete logarithm was presented by Schnorr [Sch90] and is shown in Figure 2.2. Let there be a PPT algorithm that takes as input 1^k and outputs a description $(\mathbb{G}, g, q)$ where $\mathbb{G}$ is an underlying working group, g is its generator, and q is its order. Then, the prover chooses x from $\mathbb{Z}_q$ which represents the secret key, and computes $X = g^x$. Furthermore, $(\mathbb{G}, g, q, X)$ represents the public key. Then, the prover wants to convince the verifier that he knows x and initiates the protocol by computing $V = g^v$, where $v \xleftarrow{\$} \mathbb{Z}_q$; it sends V as the initial message. Then the verifier chooses a challenge r from $\mathbb{Z}_q$ and sends it to the prover. Finally, the prover computes $z = v - rx \mod q$ and sends it to the verifier. The verifier accepts if and only if V corresponds to the $g^z X^r$.
We described an interactive scheme, where the prover convinces the verifier that he knows the discrete log x of a public element X. By applying Fiat-Shamir transformation [FS87] (challenge r is replaced by a hash function $H(\cdot)$, we obtain a non-interactive proof system that is called *The Schnorr signature scheme*, also shown in [Sch90]. Moreover, Pointcheval and Stern [PS96] established a general technique to prove the Schnorr signature scheme (Definition 2.10) secure against chosen message attacks in the random oracle model.

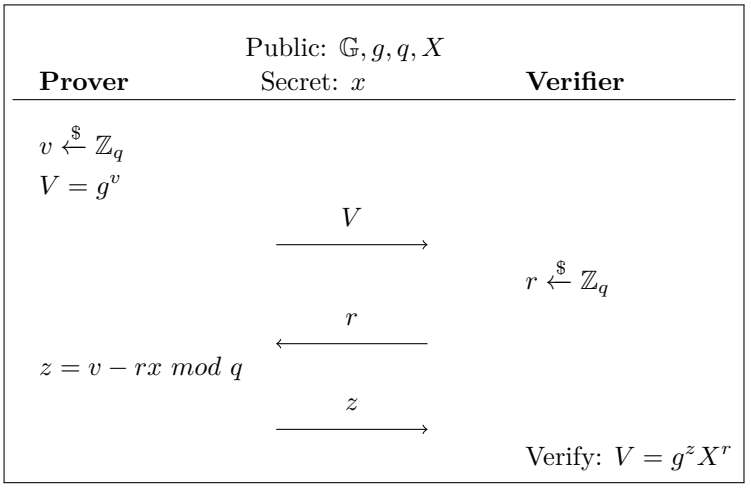

FIGURE 2.2: Schnorr Identification Scheme

Definition 2.10. The Schnorr signature scheme is a tuple of PPT algorithms *(KeyGen, Sign, Verify)* where:

- *KeyGen* is a key generation algorithm that generates a private and public key. The private key represents x which is randomly selected from $\mathbb{Z}_q$, and the public key is represented by X, computed as $X = g^x$ and $X \in \mathbb{G}$.
- *Sign* is a signing algorithm that for chosen message $m \in \{0,1\}^*$ and a label l chooses v from $\mathbb{Z}_q$ and computes (V, e) where $V = g^v$ and $e = H(m, V, l)$ and $z := v - ex \mod q$. Thus, on input (m, l), *Sign* algorithm outputs (z, e).
- *Verify* is an algorithm that verifies if the prover is being truthful by checking if $H(m, V, l) = e$ where $V = g^z X^e$. In the case of equality, *Ver* outputs 1 and 0 otherwise.

We use a non-interactive Schnorr proof of knowledge instantiated as a weaker simulation-sound extractable NIZK in the sJ-PAKE protocol in Chapter 7. This notion is shown and inherited from the original J-PAKE proof [ABM15] and we explain it in Section 2.5.3.

2.5.3 Algebraic Simulation-Sound Extractable NIZK

We substitute the simulation-sound extractability property with a weak *algebraic simulation-sound extractability* and *base indistinguishability* and we obtain a new non-interactive NIZK construction, *algebraic sound extractable NIZK*. This result was shown in [ABM15] to instantiate a non-interactive Schnorr proof of knowledge [Sch90] in the original J-PAKE protocol [HR10]. We adopt the same construction for the security proof of the sJ-PAKE protocol in Chapter 7.

The transformation of Schnorr into a SE-NIZK showed no issues for the *unbounded zero-knowledge* property as a simulated proof is easily programmable. However, there was an issue with extractability as there was no possible straight-line extractor when using Schnorr. To solve this issue, *algebraic adversaries* were introduced. Additionally, assuming the bases are hard linear and feeding the bases to the extractor is a way the extractor can extract a witness from the adversarial proof of knowledge. Now, we formally define algebraic simulation-sound extractable NIZK (alg-SE-NIZK):

Definition 2.11. Labelled *Algebraic simulation sound extractable NIZK* is a system that consists of algorithms $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver})$ for relation $\mathcal{R}$ and if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ and an extractor $\mathsf{Extract}_2$ and the following conditions hold:

1. Weak algebraic simulation-sound extractability. This property is the same as in SE-NIZK, except, the extractor $\mathsf{Extract}_2$ is given the discrete logarithms of all group elements in a base g ($g \xleftarrow{\$} G \setminus \{1\}$). This is possible because our reduction is algebraic. More formally, for all PPT adversaries $\mathcal{A}$, we define $\mathrm{Adv}^{\mathrm{weak-alg}}$:

$$\Pr\Big[(\mathsf{crs}, \mathsf{td}_s) \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa); g \xleftarrow{\$} G; (x, \pi) \leftarrow \mathcal{A}^{(g), \mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}_s, \cdot, \cdot), \mathsf{Extract}_2(\mathsf{crs}, \mathsf{td}_e, \cdot, \cdot, \cdot)}(\mathsf{crs}, \mathsf{td}_e) : \\ \mathsf{Ver}(\mathsf{crs}, x, \pi, l) = 1, ((x, l), \pi) \notin Q \wedge \mathcal{R}(x, \mathsf{Extract}_2(\mathsf{crs}, \mathsf{td}_e, x, \pi, l)) = 0\Big]$$

where Q is a set of query-response pairs $((x, l), \pi)$ for $\mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}_s, \cdot, \cdot)$.

2. Base indistinguishability. An alg-SE-NIZK has indistinguishable bases if all elements are indistinguishable when using bases ($g \xleftarrow{\$} G \setminus \{1\}$) or $(g_1, ..., g_n) \xleftarrow{\$} \mathcal{D}$. We define an advantage of any PPT adversary as $\mathrm{Adv}^{\mathrm{base-ind}}$:

$$\Pr\Big[\mathsf{crs}, \mathsf{td}_s, \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa); (g_1, ..., g_n) \xleftarrow{\$} \mathcal{D} : \\ \mathcal{A}^{(g), \mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}_s, \cdot, \cdot), \mathsf{Extract}_2(\mathsf{crs}, \mathsf{td}_e, \cdot, \cdot, \cdot)}(G, g_1, ..., g_n, \mathsf{crs}) = 1\Big] \\ - \Pr\Big[\mathsf{crs}, \mathsf{td}_s, \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa); (g_1, ..., g_n) \xleftarrow{\$} \mathcal{D} : \\ \mathcal{A}^{(g_1, ..., g_n), \mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}_s, \cdot, \cdot), \mathsf{Extract}_2(\mathsf{crs}, \mathsf{td}_e, \cdot, \cdot, \cdot)}(G, g_1, ..., g_n, \mathsf{crs}) = 1\Big]$$

where $(g_1, ...g_n)$ is a hard-linear base defined in Definition 2.12.
The resulting scheme is slightly weaker than SE-NIZK. However, the security proof for J-PAKE (and therefore for sJ-PAKE), which relies on SE-NIZK proofs, can be updated only to assume alg-SE-NIZK proofs since all reductions are algebraic and all the bases used in the proofs are hard-linear (Definition 2.12).

Definition 2.12. Given some distribution $(g_1, ..., g_n) \xleftarrow{\$} \mathcal{D}$, where $\mathcal{D}$ is a hard linear distribution of tuples in G^n, for some n, it is computationally hard to find $(\mu_1, ..., \mu_n) \neq 0$

such that $g_1^{\mu_1} \cdots g_n^{\mu_n} = 1$. More precisely, let $\text{Adv}_{\mathcal{D}}^{\text{hard}-\text{lin}}(\mathcal{A})$ be

$$\Pr\left[(g_1, ...g_n) \xleftarrow{\$} \mathcal{D}; (\mu_1, ..., \mu_n) \xleftarrow{\$} \mathcal{A}(g_1, ...g_n) : g_1^{\mu_1} \cdots g_n^{\mu_n} = 1\right]$$

For more details about alg-SE-NIZK, we refer to [ABM15].

2.6 Cryptographic Hardness Assumptions

Most cryptographic schemes cannot be proven secure unconditionally, meaning the security of a cryptographic construction in question must rely on assumptions. Assumptions are precise mathematical statements that are not proven but only conjectured to be true. For the assumption to be considered a reliable cryptographic statement, upon which we could base any security proof, it needs to be validated, studied, and tested for several years. Therefore, this approach is preferred to the one where we could simply assume that the construction itself is secure. Furthermore, when we say that the assumption is considered *hard*, it means that we have certainty that there is no PPT adversary that managed to break the assumption *so far*.

Proofs by reduction. A cryptographic scheme is considered secure if no PPT adversary can break the hard problem on which the scheme relies. More precisely, the transformation of PPT adversary that is trying to break the scheme into a PPT adversary that is trying to break a hard problem (on which given construction is based on), is called *the reduction*. Furthermore, *reducing* the probability of the adversary breaking the assumption to a probability of breaking the protocol is called *the reductionist approach*.

Tight reductions. Let us say there is a PPT adversary $\mathcal{A}$ running in time $t(k)$ with the probability success of breaking some protocol $\epsilon(k)$, for some security parameter $k \in \mathbb{N}$. Let us then say there is a reduction $\mathcal{B}^{\mathcal{A}}$ that runs $\mathcal{A}$ in time $t'(k)$, with the success probability of breaking a hard assumption (that protocol relies on) $\epsilon'(k)$. We consider the reduction to be tight if $\frac{t'(k)}{\epsilon'(k)} = c(k)\frac{t(k)}{\epsilon(k)}$, where $c(k)$ is a constant bounded by the polynomial [BJLS16]. In contrast, a loose reduction means choosing higher security parameters which can decrease the efficiency of the protocol in question. Thus, a tight reduction is preferable. Good examples of tight reductions are shown in [CGCG$^+$19] and [BIO$^+$17].

Finally, proofs by reductions are commonly used when providing security proofs of cryptographic constructions ([Mac02], [ABM15], [BOS19]). In the next sections, we list some of the standard, computational and decisional, assumptions known to be hard that we also use in our security proof (in Chapter 7). We then explain the relationships between them. We consider $\mathbb{G}$ to be a multiplicative group, with a generator g and a prime order q such that $|q| := \kappa$, for a given security parameter κ.

2.6.1 Computational Assumptions

Definition 2.13. Discrete Logarithm Problem (DLP). Given g^x, where $g^x \in \mathbb{G}$, compute x. Let the advantage of an algorithm $\mathcal{A}$ in solving the DL problem be:

$$\mathrm{Adv}^{\mathtt{DL}}_{\mathbb{G}}(\mathcal{A}) = \Pr[x \xleftarrow{\$} \mathbb{Z}_q : x = \mathcal{A}(g^x)] \tag{2.8}$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{DL}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

Definition 2.14. Computational Diffie-Hellman (CDH) Problem. Given g^x, g^y, where $\{g^x, g^y\} \in \mathbb{G}$, compute g^{xy}. Let the advantage of an algorithm $\mathcal{A}$ in solving the CDH problem be:

$$\mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[x, y \xleftarrow{\$} \mathbb{Z}_q : g^{xy} = \mathcal{A}(g^x, g^y)\right]. \tag{2.9}$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

Definition 2.15. Computational Square Diffie-Hellman (CqSDH) Problem. Given g^x, where $\{g^x, g^{x^2}\} \in \mathbb{G}$, compute g^{x^2}. Let the advantage of an algorithm $\mathcal{A}$ in solving the CSqDH problem be:

$$\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[x \xleftarrow{\$} \mathbb{Z}_q : g^{x^2} = \mathcal{A}(g^x)\right]. \tag{2.10}$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

Definition 2.16. Computational Triple Group Diffie-Hellman (CTGDH) Problem. Given g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, where $\{g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^{xyz}\} \in \mathbb{G}$ and $g^{xy} = DH(g^x, g^y)$, $g^{xz} = DH(g^x, g^z)$, $g^{yz} = DH(g^y, g^z)$, compute g^{xyz}. Let the advantage of an algorithm $\mathcal{A}$ in solving the CTGDH problem be:

$$\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[(x, y, z) \xleftarrow{\$} \mathbb{Z}_q^3 : g^{xyz} = \mathcal{A}(g^x, g^y, g^z, g^{xy}, g^{yz}, g^{xz})\right]. \tag{2.11}$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

2.6.2 Decisional Assumptions

Definition 2.17. Decisional Diffie-Hellman (DDH) Problem. Given $(g^x, g^y, g^z) \in \mathbb{G}$, distinguish whether $z = xy$ or $z = r$, where r is is randomly chosen from $\mathbb{Z}_q$. Let the advantage of algorithm $\mathcal{A}$ in solving the DDH problem be:

$$\mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{A}) = |\Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2 : 1 = \mathcal{A}(g^x, g^y, g^{xy})\right] - \Pr\left[(x, y, r) \xleftarrow{\$} \mathbb{Z}_q^3 : 1 = \mathcal{A}(g^x, g^y, g^r)\right]|.$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

Definition 2.18. Decisional Square Diffie-Hellman (DSqDH) Problem. Given (g^x, g^z), distinguish whether $z = x^2$ or $z = r$, where r is randomly chosen from $\mathbb{Z}_q$ and $\{g^x, g^z\} \in \mathbb{G}$. Let the advantage of algorithm $\mathcal{A}$ in solving the DSqDH problem be:

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{DSqDH}}(\mathcal{A}) = |\Pr\left[x \xleftarrow{\$} \mathbb{Z}_q : 1 = \mathcal{A}(g^x, g^{x^2})\right] - \Pr\left[(x, r) \xleftarrow{\$} \mathbb{Z}_q^2 : 1 = \mathcal{A}(g^x, g^r)\right]|. \quad (2.12)$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{DSqDH}}(\mathcal{A})$ is a negligible function.

Definition 2.19. Decisional Triple Group Diffie-Hellman (DTGDH) Problem. Given $g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^w$ where $\{g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^w\} \in \mathbb{G}$ and $g^{xy} = DH(g^x, g^y)$, $g^{xz} = DH(g^x, g^z)$, $g^{yz} = DH(g^y, g^z)$, distinguish whether $w = xyz$ or $w = r$, where r is randomly chosen from $\mathbb{Z}_q$. Let the advantage of an algorithm $\mathcal{A}$ in solving the DTGDH problem be:

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{DTGDH}}(\mathcal{A}) = |\Pr\left[x, y, z \xleftarrow{\$} \mathbb{Z}_q : 1 = \mathcal{A}(g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^{xyz})\right] - \Pr\left[(x, y, z, r) \xleftarrow{\$} \mathbb{Z}_q^2 : 1 = \mathcal{A}g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^r)\right]|. \quad (2.13)$$

For all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{DTGDH}}(\mathcal{A})$ is a negligible function.

2.6.3 Relations Between the Assumptions

A security proof is more meaningful when it relies on a weaker assumption, i.e., implied by the stronger assumption. We layout the relationships of known assumptions in Figure 2.3. First, we ordered the assumptions from the well-known to least known (from left to right), and we used the same order to imply the strength of the assumption (for example, DSqDH is the strongest assumption on the list). Second, the computational assumptions are considered to be weaker than decisional assumptions. Finally, we base our analysis on the relations between the assumptions from Figure 2.3 on [ABM15], [BDZ03] and [STW96]. For the sake of clarity, we provide relations as separate statements below[5]. For all PPT algorithms $\mathcal{B}$ and $\mathcal{B}^{\mathcal{A}}$, running in time t:

- $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CSqDH}}(\mathcal{B}) \geq \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{B}^{\mathcal{A}})$
- $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{B}) \geq (\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CSqDH}}(\mathcal{B}^{\mathcal{A}}))^2$
- $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CTGDH}}(\mathcal{B}) \geq \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{B}^{\mathcal{A}})$
- $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CSqDH}}(\mathcal{B}) \geq \mathrm{Adv}_{\mathbb{G}}^{\mathtt{DL}}(\mathcal{B}^{\mathcal{A}})$

[5]We simplify our analysis by ignoring the time of exponentiation (t_{exp})

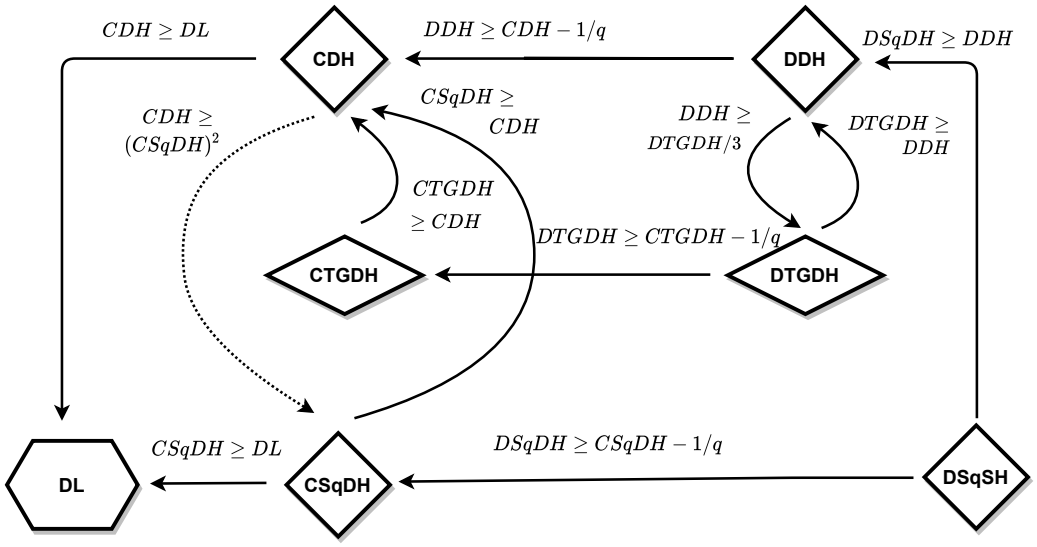

FIGURE 2.3: Relationships between the assumptions also shown in [ABM15]

.

- $\mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{DL}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}})$
- $\mathrm{Adv}^{\mathtt{DSqDH}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}})$
- $\mathrm{Adv}^{\mathtt{DSqDH}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}}) - 1/q$
- $\mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}}) - 1/q$
- $\mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{B}) \geq (\mathrm{Adv}^{\mathtt{DTGDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}}))/3$
- $\mathrm{Adv}^{\mathtt{DTGDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}})$
- $\mathrm{Adv}^{\mathtt{DTGDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{CTGDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}}) - 1/q$

In the sciences, the authority of thousands of opinions is not worth as much as one tiny spark of reason in an individual man.

Galileo Galilei

3

Password Authenticated Key-Exchange Protocols

3.1 Introduction

The problem of finding a robust solution for secure client authentication on the Internet is of extraordinary importance, especially today when we are facing a tremendous increase in the number of IoT-enabled devices and third-party services that require user authentication[1]. The potential risk in terms of breaching users' security and privacy posed by such development coupled with the amount of content and data generated and stored in cloud storage can only be imagined. Even though some predicted that the password would die out, they are still the most widely used tool for authentication. To maliciously authenticate into password-protected accounts, one needs to perform so-called *password attacks*. These attacks are typically performed using software that facilitates cracking or guessing passwords. The question of how the password is stored on the server raised serious issues when server compromises occurred in Linkedin [Lin12] in 2012, Facebook in 2019 [Fac19], and the most recent Paxful [Pax21] trading platform in 2021. An adversary can exploit the fact that users re-use the same passwords on several services. This means that even when the password of the leaked server is changed, an attacker can still perform a *credential stuffing attack* where he attempts to log in with

[1]Most estimates agree that the number of existing IoT-enabled devices surpassed the number of people in the world.

already breached passwords into other uncompromised services, increasing his chances of successfully impersonating the user. In this chapter, we present Password Authenticated Key-Exchange (PAKE) primitive, which, if properly designed, can mitigate some of the issues mentioned above in certain use cases. This would include protection against offline dictionary attacks, phishing, and future password leaks, among others.

3.2 Password Authenticated Key-Exchange Protocols

The idea of bootstrapping a low-entropy secret into high-entropy session keys first appeared in [Mer82]. Protocols supporting this idea and providing authenticity over an insecure channel are called Password Authenticated Key Exchange protocols (PAKE). The key challenge in designing a PAKE is to ensure that an attacker cannot derive sufficient information to execute an offline dictionary attack, i.e. never gets enough information to (tractably) confirm or deny in an offline computation a guess at the password. Such attackers might be passive, simply eavesdropping, or active, masquerading as one or more parties. These two attack models are incomparable. This is essential as the passwords are typically low entropy and hence vulnerable to brute force attack. Of course, online guesses are unavoidable, but such attacks are detectable by the honest parties and can be throttled, by for example, limiting the number of tries. Furthermore, PAKE is a protocol executed between two honest parties in a concurrent setting, meaning each participant is permitted to have *multiple instantiations* simultaneously. Thus, to successfully run *multiple sessions*, the following properties need to be fulfilled [Kra03].

Key Authentication. When two honest users start communicating, they establish a *session*. At the end of each session, each party should be able to verify the other party's identity, *pid*, and both parties should hold the same session key, *sk*. Furthermore, the session id, *sid*, should be *unique* to each pair of users. PAKEs offer two kinds of authentication mechanisms:

- Implicit key authentication. After the protocol run is completed, two parties hold a session key *sk*. Implicit authentication *guarantees* that only parties who complete the protocol session can obtain the same session key (on both sides). Conversely, if a party has not completed the protocol with its desired partner, then the partner will not hold the same session key as the party. In addition, the execution of the protocol usually continues until the end-if all the checks during the protocol are verified. Some of the PAKE protocols with implicit authentication are [HR10], [AFP05].

- Explicit key authentication. After the protocol have been executed successfully, both users are certain that the other said is who they say they are. The certainty

is usually achieved with *confirmation codes*. However, the protocol does not necessarily run to the end, meaning that if the confirmation code is in the middle of the execution as in [Mac02] and the confirmation code does not verify, the protocol aborts.

Consistency. After A and B execute the protocol, A outputs $(sk_A, sid_A, pid_A = B)$ and B outputs $(sk_B, sid_B, pid_B = A)$. Both outputs need to have a consistent view of who their partner is for that particular session.

Secrecy. No third party should learn anything about the session key, whether by observing or interfering with the protocol established by two honest parties. Furthermore, the session key should be indistinguishable from any random session key.

3.2.1 Passwords and Session Keys

Passwords are a unique set of characters, conveniently used as a login tool for every person. Key-exchange protocols that use passwords as a means for authentication provide two main advantages:

- User friendly. Every person can simply choose any password he wants and associate it with his desired internet or bank activity. Passwords are easy to remember and, if compromised, easy to replace, making them the most widely used authentication tool.
- No Public Key Infrastructure (PKI). Sourcing and policy drafting are just some of the things that make managing certificates costly and time-consuming. Fortunately, PAKEs solely fuel on passwords and do not rely on PKI.

According to Social Media Benchmark report in 2020 [Omn20], each user on average has 8 active social media accounts in the USA. One can assume that this number is much higher, considering accounts on other applications. The high number of accounts can somehow affect the quality of passwords. More precisely, users often use the same or correlated passwords as part of the login credentials for more than one account. This means, if the adversary cracks the password for one account, he can gain access to other accounts associated with the same user. Another feature that makes PAKE a strong cryptographic mechanism is that it delivers strong session keys that can be subsequently used in the encryption process of some data. To ensure that data is encrypted using strong session keys, the following properties need to be fulfilled:

- Every communication should be established with a different session key;

- Every communication should have an independent session key, meaning, the keys should not be correlated with any keys (established in previous sessions);
- All session keys should be deleted after the corresponding session expires;

In some cases, PAKE protocols are used just for authentication [1Pa18], making them an over-kill as the session key agreed at the end of the protocol is not subsequently used. Instead, one could consider using passwords directly as long-term secrets for encryption. However, there are a couple of downsides to that approach:

- Using passwords for encryption can lead to partition attacks, as in EKE [BR93].
- One should avoid using the same session key more than once to prevent crypto-analysis on ciphertexts.
- If there is a password leak at any point in the protocol, whether the adversary is actively participating or just observing, the protection of session keys in previous sessions is not guaranteed, i.e. *forward secrecy* property is not satisfied.

Overall, the simplicity of password-based, free of PKI and forward secrecy, make PAKE protocols unique and appealing for wider adoption.

3.2.2 Attacks on PAKE Protocols

The fact that passwords are still the most widely used authentication mechanism certainly raises interest in using PAKEs. However, because of the easy-to-guess nature of passwords [YBAG04], the following attacks are possible:

Dictionary attacks. Dictionaries are big sets of the most used human-memorable passwords, that adversaries use to perform *dictionary attacks*. Depending on whether the attack takes place *online* or *offline* we can distinguish:

- Offline dictionary attack is when an adversary obtains information about the password *without any time limitation* and *online connection*. When building a dictionary, an adversary can use it as a *verifier* when performing an exhaustive password search. If the information he obtained matches the verifier with the particular password input, then the adversary successfully cracked the password, i.e. retrieved the password *in the clear*. Some of the already existing dictionaries are HashCat [Has19] and John the Ripper [joh19]. Our first work in Chapter 5 shows a real-world example of a PAKE protocol prone to offline dictionary attacks and displays an empirical analysis of the attack.
- An online dictionary attack is an attack where the adversary *interacts* with the website, trying to impersonate the user by simply inputting the candidate password

from the dictionary when running the key-exchange protocol. The protocol verifies whether the password is correct in which case it leads to successful login.

The minimum requirement for security of PAKE protocols is that it is resistant to offline dictionary attacks and that the adversary is confined to only online dictionary attacks. More precisely, when performing an online dictionary attack, the attacker is allowed to test only one password per session.
Another type of attack that captures the scenarios where online guessing is misinterpreted for network failures is also possible with PAKEs. Furthermore, Roscoe and Ryan [Rya17] showed that PAKEs could fulfill *auditability*[2] property by developing a new stochastic approximation to *fair* information exchange between the honest parties.

Pre-computation attacks. In the *client-server* setting, passwords are stored on a server-side in the form of a *hash* instead of *in clear*. Even though in this way passwords are considered to be *safer*, protocols usually use a deterministic password mapping[3] which then attracts the adversary to do computations on passwords beforehand. More precisely, the adversary can compute a table of values based on a passwords dictionary before comprising a server, which *may* lead to an *immediate* password recovery once the server is compromised. For instance, the adversary computes $H(\pi_i)$, where $\{\pi_1, \pi_2, ..., \pi_n\}$ come from a dictionary and builds a *hash table*. Afterward, the adversary compromises the server and compares it with a pre-computed table, and in case of a match, he rapidly retrieves the password. A possible PAKE solution that is resistant to pre-computation attacks is OPAQUE [JKSS18] which we describe in detail below (in Section 3.2.4).

3.2.3 Other Password-based Authentication

One of the reasons why online dictionary attacks are not problematic is that they are usually limited to just a couple of login attempts, which seriously mitigates the successful login entry. Another reason is that the websites usually instruct their users to choose strong passwords, even hinting at the number and the nature of characters. These reasons suggest that the adversary needs to be very lucky to perform an online dictionary attack successfully. Because passwords are a minimum requirement for login accounts, the most typical authentication is password-based, either on the client-side or the server-side. Some of the other authentication password-based mechanisms worth mentioning are:

[2] A protocol is considered to be *auditable* if honest parties can distinguish between online guessing and a communication failure.

[3] The protocols can also use random salt, transmitted in the clear from servers to users.

- Two-factor authentication (TFA), where, besides inputting the password, the user must provide a digit code chosen from an auxiliary device (e.g., a smartphone or a USB token).

- Two-factor PAKE with end-to-end security, where, besides the password, the user uses any device-enhanced PAKE (defined by Jarecki et al. [JKSS18]) and any Short-Authenticated-String Message Authentication (SAS-MA) (defined by Vaudenay [Vau05]). This modular construction is efficiently instantiated and is a part of recent work from Jarecki et al. [JJK+21]. Furthermore, it achieves security against any attacker, passive or active, attacking any part of the system.

- aPAKE-into-TLS login, described in [SKFB21], is an approach that first establishes a TLS connection between the user and the server and then uses aPAKE to preserve the secrecy of the user's credentials. However, aPAKE-into-TLS is not always the desired login, as it requires merging the application and the network layer to integrate aPAKE into TLS successfully. For this reason, Lewi, Mohassel and Roy [LMR20] introduce a new primitive, credential-hiding login (CHL), which is more efficient than aPAKE-into-TLS while preserving the same security[4] as aPAKE-into-TLS approach.

- Given one master password, password managers create and store multiple high-entropy passwords based on the master password [SJKS17].

- An interesting mechanism called *HoneyPAKE*, based on Juels and Rivest [JR13], integrates PAKE with another auxiliary device called the HoneyChecker, which gets triggered, whenever the adversary may attempt to compromise a server. HoneyPAKEs are considered to add another layer of protection to passwords and are explained in detail, in our second work in Chapter 6.

- *Oblivious PAKE* (O-PAKE) [KM13] is an interesting PAKE construction where a client has a set of passwords, among which, *only one* is shared with a server. The interesting part is that no information about any password is leaked and a server uses a special technique called *Index Hiding Message Encoding* (IHME) to retrieve the password. The goal of O-PAKE is to improve the efficiency of PAKE protocols in the scenarios where the client mistypes the password or can not remember the exact password being used for a particular website.

3.2.4 Balanced and Augmented PAKEs

So far, two types of PAKE protocols have been proposed, depending on how the password is stored on the server: *cleartext* or in the form of a *hash*. PAKE protocols where

[4] Universal Composability framework [Can01]

the server stores a password in clear text are called *balanced* and *augmented* otherwise. Unlike balanced PAKEs, augmented PAKEs are more resistant to server compromise. Furthermore, if implemented well, clients' passwords are not immediately revealed once the password file is leaked since the attacker still has to perform password cracking. Unfortunately, augmented PAKEs only make it somewhat difficult for the adversary to retrieve the passwords, but they do not remedy pre-computation attacks.

Resilience to pre-computation attacks. Despite the plethora of PAKE protocols proposed in the last 25 years, almost all of them are vulnerable to already mentioned pre-computation attacks. However, the OPAQUE protocol [JKSS18], proposed in 2018 stands out with exactly that property. Pre-computation attacks are possible if the server, in a *client-server* setting, sends the salt *in the clear* to the client, making it possible for the adversary to pre-compute an offline dictionary and thus do the pre-computation attack. In most of the PAKE protocols in the Initialization phase, a client receives a salt in the clear from the server, making it possible for the adversary to impersonate the client and perform pre-computation attacks. In contrast, OPAQUE gets around the pre-computation attacks in the following way:

- It does not reveal the salt to the attacker. Furthermore, it uses an efficient oblivious pseudorandom function (OPRF) to combine the salt with the password, ensuring that the client does not learn the salt and the server does not learn the password.
- It is possible to take all the hashing work from the *the server-side* and given to the *client* side, freeing the server from a workload so it can use stronger security settings like hard key derivation function, for instance, *scrypt*, with large RAM parameters[5], that in the end, significantly slows down (pre)computation process.

In conclusion, the OPAQUE protocol is a sort of new breed of PAKE protocols, called *strong augmented PAKE*[6], which may take over the reins when the future of the password-based protocols is considered.

3.3 Security properties in PAKE Protocols

According to [HR10], every PAKE should fulfil the following requirements:

Resistance to Eavesdropping. An adversary should not obtain any information that would aid him to retrieve the password when observing the protocol execution.

[5]Standard scrypt parameters are $N = 16384$, $r = 8$, $p = 1$ [scr]
[6]Sometimes also referred to as asymmetric PAKE

Offline Dictionary Attack Resistance. An adversary should not be able to obtain any useful information on which he could perform an offline dictionary attack. A clear example of an unsafe PAKE protocol where an adversary can perform an offline dictionary attack is zkPAKE [MRA15] (covered in Chapter 5).

Bounded Online Dictionary Attacks. An adversary should be limited to one password *guess* per active session.

Forward Secrecy. One of the benefits of PAKE protocols is achieving forward secrecy. This means that if a long-term secret (i.e. password in case of PAKE) gets compromised, the session keys remain protected. Depending on the participation of the adversary, there are two levels of forward secrecy.

- Weak forward secrecy (wFS): Session keys are protected after long-term secret is compromised when the adversary is just eavesdropping.
- Perfect forward secrecy (PFS): Session keys from previous sessions where the adversary was *actively interfering* (e.g. sending or modifying messages), are protected if a long-term secret gets compromised.

Perfect forward secrecy is a stronger notion [Kra05]. Moreover, it is required for any PAKE protocol to be seriously taken under consideration to be deployed. We define more formally the conditions in the indistinguishability-based models for perfect-forward secrecy (PFS) and weak-forward secrecy (wFS) in Chapter 4.

3.4 Previous Works

Although there has been an abundance of PAKE protocols, see [BM92, Wu98, Jab96, JR13, KOY01, Har15, AP05, HR10, TWMP07, BR94, Mac02, BRRS18, HL19, KM13], in the last two and a half decades, we mention just a representative few relevant to our work. For a more detailed overview of PAKE research field, we refer to Pointcheval's survey [Poi12]. We start with EKE by Bellovin and Merritt [BM92] and SPEKE [Jab96] which were seminal but shown prone to attacks. The popularity of PAKE protocols grew as prominent complexity-theoretic security models ([BPR00, AFP05, BMP00, CK01]) precisely defined guidelines to prove PAKE protocols secure. For instance, PAK and PPK [Mac02], SPAKE2 [AP05], J-PAKE [HR10] are proven secure in the *game-based models* [AFP05, BPR00]. Even though game-based models are considered to be the weakest security proofs models for PAKE(s), they still represent a high-standard bar that a certain PAKE protocol needs to reach to be taken into consideration for practice use. Unlike the game-based models, a model that makes no assumptions on the passwords and captures more real-world adversaries is the *Universally Composable* (UC) model designed

by Canetti et al. [CHK+05]. Protocols like, SPAKE2+ [Sho20], OPAQUE [JKX18] and CPace ([HL18], [AHH21]), are some of the more recent ones that are proven secure in the UC framework, guaranteeing security under arbitrary protocol composition. In the latest work, Gu, Jarecki and Krawczyk introduce the KHAPKE protocol [GJK21], which is an improved version of OPAQUE that does not rely on the security of the oblivious pseudo-random function (OPRF), also proven in the UC setting. Until recently, only protocols that could administrate the scenario where an adversary conducts an online attack against the session by making a password guess *before* that session is completed could be proven UC secure. Because of this property, some of the protocols ([AP05], [HS14, Jab97, Mac01], [PW17]) could only be proven secure in the game-based setting. Later, Abdalla et al. [ABB+20] introduced the relaxation of the PAKE functionality in UC, *lazy-extraction PAKE* (lePAKE) by allowing the ideal-world adversary to postpone the password guess even after the session ends. This relaxation allowed the SPAKE2 [AP05], SPEKE [HS14, Jab97, Mac01], and TBSPEKE [PW17] to be proven in relaxed UC. PAKEs also found their use in PakeMail [SALR21], where they simplify entity authentication in decentralized, secure email by not relying on PKI nor trusted setup. Lastly, a new notion of public-key encryption, *Password-based Authenticated Public-key Encryption* (PAPKE), was introduced by Jarecki et al. [BCJ+19]. To better understand this primitive, the authors related it to PAKEs, first, by showing that the PAPKE protocol strictly implies PAKE. Second, their work shows that any generic PAPKE-to-PAKE compiler builds the two-round UC PAKE from UC PAPKE. Finally, looking back on the more recent previous work on PAKEs and its latest IETF standardization, it seems that the UC framework has become a standard security model for PAKEs.

3.5 Applications of PAKE protocols

While security proofs are desirable, they are not vital for the protocol to be deployed in practice. A prime example of that is SRP [Wu98] which is one of the most used PAKEs due to its implementation compatible nature. Next, we list the concrete applications and name the protocols they adopted.

- Password manager called 1Password [1Pa18] uses SRP for authentication.
- Apple iCloud Keychain allows users to synchronize their passwords between iCloud-enabled devices securely. In addition, the SRP is used as an authentication mechanism during the recovery process: the user must prove knowledge of the previously registered Passcode without revealing it to anyone – not even Apple.

- Wi-Fi Protected Access v3 (WPA3) is the standard for secure wireless communications. It adopted Dragonfly protocol [Har15, Lv15] for authentication to prevent offline dictionary attacks when a pre-shared key has been leaked.

- The SRP PAKE protocol has been standardized as a TLS *cipher suite* [TWMP07]. The reason that it failed to be widely adopted [MSKD16] in the TLS cipher suite is: i) the need for significant modifications to the TLS *handshake* protocol, ii) required changes in the web browsers, and iii) abandoning the public-key infrastructure. A better approach was proposed in [EKSS09, MSKD16], which keeps TLS intact and only affects the design of the web browser, particularly in the way users input their credentials. However, for the lack of concrete user-experience studies, this approach still needs to be confirmed.

- The International Civil Aviation Organization (ICAO) uses PACE protocol [BK09] to protect the data stored in the chip from unauthorized access and establish a secure channel between the chip and the passport reader.

- Amazon Cognito is an authentication module developed by Amazon Web Services [Ama20], which handles user registration, authentication, and account recovery. It uses the SRP protocol for authentication.

- Magic Wormhole is a software that allows two users to rapidly exchange big files (ex. video) to avoid trusting the server that uploaded files will not be compromised. This software uses SPAKE2 [AP05] for authentication.

One of the most widely deployed PAKEs, is J-PAKE [HR10]. Even though J-PAKE is already mentioned above, some of the other applications that adopted J-PAKE are Thread protocol (in IoT) [Thr16], Pale Moon [Pal16] and OpenSSL [Ope16] library.

3.6 IETF Standardization

As already mentioned in Section 1.3.2, the winning protocols of the last IETF competition were CPace ([HL18], [AHH21]) and OPAQUE [JKX18]. Unfortunately, one of the candidates, J-PAKE, was removed in the early stages of the competition for not fulfilling the "one round" efficiency requirement. However, we believe there are reasons why J-PAKE is still an important protocol to consider. One reason is that, compared to the winning CPace, it does not require hashing into a curve which might cause timing attacks [VR19]. Another one is that J-PAKE uses short exponents in contrast to CPace and OPAQUE, making one modular exponentiation in CPace and OPAQUE around 9 times costlier than in J-PAKE. For more technical details, we refer to [Hao21]. Finally, the third work (Chapter 7) in this dissertation presents and proves the sJ-PAKE protocol

as a lighter and more efficient version of J-PAKE, which shows that J-PAKE protocol still has a lot to offer.

The greatest enemy of knowledge is not ignorance; it is the illusion of knowledge.

Stephen Hawking

4

Security Models for Password Authenticated Key-Exchange Protocols

When designing PAKE protocols, one must perform rigorous analysis of its security and efficiency. Real-world examples, such as SRP [Wu98], [1] have shown that a security proof is not vital if there are no known attacks. However, providing the security proof is a necessary task because of the following reasons: (*i*) to avoid publishing protocols that are afterwards shown to be broken [Szy06], [BŠŠ17], [CH14], [MPS00]; (*ii*) for a protocol to be considered as a candidate for any type of standardization (IETF, IEEE, ISO); (*iii*) for the wide adoption in TLS or some other mechanism. Based on the works of Bellare and Rogaway ([BR93], [BR95], [BM97]), the first proposed password authenticated key-exchange protocols complexity-theoretical security models are indistinguishability-based (IND-based) game-based, so-called *Find-then-Guess* [BPR00] and *Real-or-Random* [AP05], where the security protocol is considered to be breached if the adversary can *distinguish* between the *real* and *random* session keys. Then, a *simulation-based* (SIM-based) model captures an *ideal* world and a *real* world protocol. The security holds if the adversary cannot distinguish whether it interacts with the real or with an idealized protocol that is secure by definition. One example of the SIM-based protocol is based on Shoup's work on authenticated key exchange [Sho99], [BMP00]. Another two examples of the SIM-based model is the UC security model ([CHK$^+$05], [Can01]) and the relaxed UC model [ABB$^+$20]. SIM-based models are considered to be

[1] Currently, v6a is in use.

stronger models than IND-based, due to more realistic assumptions on passwords and capturing composability properties. The main reason supporting this statement is that IND-based models mostly consider password distribution to be uniform, which is less realistic than the passwords coming from a highly *non-uniform* distribution, modeled in simulation-based models. The second reason is that when a certain protocol is secure in the simulation-based model (e.g. UC), it implies that the same security is preserved when composed with another protocol, namely *encryption*. However, Skrobot and Lancrenon [SL18] showed that, under certain conditions, the RoR model could be extended to capture security guarantees when PAKE is composed with a different protocol.

4.1 Indistinguishability-based Models

Both, Real-or-Random (RoR) [AP05] and Find-then-Guess (FtG) [BPR00], models define the security via games, between an adversary $\mathcal{A}$ and the challenger $\mathcal{CH}$, where the adversary targets the session in which he might be able to distinguish between the real and random session keys. More precisely, the security captures the adversary's abilities to guess the bit $b \in \{0, 1\}$ which represents a real or random session key. In RoR, bit b is chosen at the beginning of the game, and $\mathcal{A}$ may guess the bit b on *multiple* instances, while in FtG, $\mathcal{A}$ may guess the bit b on *only one* instance in the security experiment. For this reason, RoR is considered to be a stronger flavor than the FtG model. For comparison and deeper analysis of both models, we refer to [AP05]. The proof in both models follows standard *theoretic-complexity* methodology [GM84], where the advantage of the adversary is bounded by the success probability of $\mathcal{A}$ solving some hard problem, i.e. the advantage of the adversary to break the protocol is *reduced* to a hard problem.

Security reductions. Once we precisely define security notions we wish to achieve for a particular scheme, we must choose a suitable security model to achieve a concrete security goal. To exhibit the security of the protocol in question, we need to demonstrate that the underlying protocol is hard to break only if the underlying assumption is believed to be hard. Thus, we need to *reduce* the security of the scheme to a security of the underlying assumption. We call this approach the *reductionist approach*. We use the reductionist approach in the security proof in our final work in Chapter 7.

4.2 Proof by Sequence of Games

As already mentioned, security proofs are vital when a new protocol is considered for real-world deployment. Security proofs are usually complex and hard to understand, so, usually, it is a good idea to break it down into smaller, less complex parts. Furthermore,

one needs remember that the proof needs to be a mirror of clarity and straightforward analysis. In addition, it needs to represent a unity of reasonable mathematical deductions and well defined claims. In this section, we explain a well-known *game-based technique* for organizing proofs, demonstrated in [Sho04], that is already widely adopted when considering (P)AKE protocols. Since we organize a complex proof into game-based sequence, we have to make sure that the games capture small changes so the transition between them is "smoother" and straightforward. We compare the probabilities of every two successive games and show that their difference is less than negligible. We recognize three different types of games that security proof usually contains.

Transitions based on indistinguishability. When comparing two successive games, sometimes, we need to base the games differences on *indistinguishability*. More precisely, two games are assumed to be indistinguishable if there exists a *distinguisher* that, given two distributions (one assigned to Game i and the other to Game $i+1$), it "interpolates" between Game i and Game $i+1$. Typically, one designs the games so that it is possible for the distinguisher to successfully interpolate, i.e. output 1 if the event that the game is trying to simulate occurs. Finally, we give an example of such a distinguisher in Game 4.5 in our proof of sJ-PAKE in Chapter 7, Section 7.4, where we have a DDH distinguisher which, given (g^x, g^y, g^z) plays Game i, if $z = xy$, and plays Game $i+1$, if z is random.

Bridging step. This type of transition is usually used to simplify the transition from Game i to Game $i+1$, when there is no bad event nor indistinguishability based difference. More precisely, bridging steps are used when certain values need to be reset or redefined equivalently. Furthermore, this type makes proof easier to follow.
We stress that one game may contain more games of different types. One such example is Game 8 (Chapter 7, Section 7.4).

Transition based on failure events. Another interesting type, commonly known to provers, is when two games are identical unless a so-called *bad* (failure) event occurs. The bad event refers to a certain random variable that is computed differently in Game i than in Game $i+1$. Furthermore, using the *Difference Lemma* below (see Definition 4.1), we can bound the winning probability of the adversary in each game hop. Finally, we give an example of a game based on failure events in Game 6 (Chapter 7, Section 7.4).

Definition 4.1. (Difference Lemma) Let G, H, F be events defined in some probabilistic distribution and suppose that $G \wedge \neg F \iff H \wedge \neg F$. Then $|\Pr[G] - \Pr[H]| \leq \Pr[F]$

Proof and more details are shown in [Sho04].

4.3 Indistinguishability-based Real-or-Random Model

In this section, we provide more details on the indistinguishability-based, Real-or-Random (RoR) model [AP05] that we use to prove the security of the sJ-PAKE protocol in Chapter 7. In RoR, adversary $\mathcal{A}$ has complete control of the network and engages in concurrent executions by *actively* participating or *passively* observing.[2] Furthermore, it guarantees the forward secrecy and incorporates all the requirements that PAKE must fulfil: (*i*) $\mathcal{A}$ can verify at most one password per session; (*ii*) $\mathcal{A}$ cannot learn anything about the password when just observing the protocol.[3] For the security experiment to work, the games need to be administrated consistently in $\mathcal{A}$'s view. We say that the adversary *wins* the game if he can distinguish *real* session keys from *random* strings. More precisely, $\mathcal{A}$'s goal is to guess the bit b, chosen at the beginning of each game, representing a real or random session key.

Password distribution. One of the downsides of indistinguishability-based models is that they assume that passwords are independent and uniformly distributed. We can extend the proof model by assuming that the passwords have minimal entropy. However, it is not realistic to assume that passwords are not related and independent. Users usually use the same or correlated passwords across multiple websites to make their life easier. For this reason, simulation-based models are a preference.

Participants and passwords. Each participant, denoted Π_U^i, is either a client $C \in \mathcal{C}$ or a server $S \in \mathcal{S}$. Each client C holds a password pw_C and the server S holds a vector of passwords $\mathsf{pw}_S =< \mathsf{pw}_C >_{C \in \mathcal{C}}$, such that $\mathsf{pw}_S[C] = \mathsf{pw}_C$ for all $C \in \mathcal{C}$. For simplicity, in our simulation we assume that passwords are independent and uniformly distributed, drawn from the password dictionary.

User instances. Each user is allowed to run simultaneously more than one protocol execution, which is modelled by allowing each user an unlimited number of *instances*. Specifically, let Π_C^i and Π_S^j denote the i-th and j-th instance of client C and server S respectively.

Protocol execution and initialization phase. The protocol P is an algorithm that defines how users respond to messages from their environment. We allow each instance to execute P unlimited times with as many other instances as it wishes. Formally, a bit is flipped at the beginning of the protocol. Then, we introduce a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$, that has full control of the network and communicates with

[2] We also consider the adversary who only forwards the messages to be passive.
[3] Just forwarding messages is considered a stronger notion than just observing the protocol.

each instance according to the rules of P via the following queries:

Send(U, i, m): A message m is sent to instance Π_U^i, which proceeds according to P and its response – if any – is given to $\mathcal{A}$. This query models an active adversary. Furthermore, $\mathcal{A}$ is notified about any state change between the instances, such as: (i) *reject*: the reason why an instance might reject is if its partner sent a message that requires verification and fails for some reason. This usually happens when confirmation codes are not matching or some protocol requirement is not fulfilled. (ii) *terminate*: An instance terminates if it has a *partner* and holds the session id (sid), partner id (pid) and a session key (sk). The *terminate* state is usually important for the adversary because instances that terminate are considered to be *fresh*. Therefore, to maintain *fair play* in the game, $\mathcal{A}$ is allowed to ask Reveal or Test query only to fresh instances.

Execute(C, i, S, j): This query triggers an honest run of protocol P between Π_C^i and Π_S^j. The transcript of the execution is given to $\mathcal{A}$. This query models an adversary who is just observing an honest protocol execution. However, the adversary is also considered to be passive when just forwarding the messages. The difference between the two scenarios is in the amount of Send queries when bounding the online dictionary attacks. This means that the non-negligible part in the bound (Equation 4.2) will linearly increase when the adversary just forwards the messages.

Corrupt(C, S): $\mathcal{A}$ receives a password pw_{cs} shared between a pair of instances, Π_C^i and Π_S^j. In a real-world scenario, this query models the compromise of the long-term secret due to security violation. More precisely, the adversary can get a hold of the password in various ways, such as compromising some system or simply performing a phishing attack on a targeted user.

Reveal(U, i): When $\mathcal{A}$ triggers this query, it receives a session key held by Π_U^i. For this query to be valid, an instance must reach the state where it computed the session key; otherwise, $\mathcal{A}$ receives a response marked as "Invalid" ($\perp$). This query models a potential session session key leakage in the real world because of its use in higher-level protocols.

Test(U, i): At the beginning of the experiment, the challenger flips a coin and sets a bit b outside of $\mathcal{A}$'s view. Then, whenever $\mathcal{A}$ asks this query to some instance Π_U^i, it obtains the following response:

1. If $b = 1$, $\mathcal{A}$ gets the real session key sk_U^i.

2. Otherwise, $\mathcal{A}$ gets a random string $r \xleftarrow{\$} \{0,1\}^\kappa$. For consistency, if two partnered instances, Π^i_C and Π^j_S, receive a Test query, then $\mathcal{A}$ gets the same random string.

The adversary can ask this query on *multiple fresh* instances. Furthermore, we only model adversarial guess success with this query, which does not correspond to any real-world scenario.

Partnered instances. Two instances, Π^i_C and Π^j_S are *partnered* if both hold the same partner identity (pid), transcript (sid) and the same session key (sk) and do not expect any messages to come their way. More precisely, a client with $(\mathsf{pid}^i_C, \mathsf{sid}^i_C, \mathsf{sk}^i_C)$ and a server with $(\mathsf{pid}^j_S, \mathsf{sid}^j_S, \mathsf{sk}^j_S)$ are partnered if:

1. $\mathsf{sid}^i_C = \mathsf{sid}^j_S$, $\mathsf{sk}^i_C = \mathsf{sk}^j_S$, $\mathsf{pid}^i_C{=}S$, $\mathsf{pid}^j_S{=}C$.
2. No other instance accepts with the same sid, except with the negligible probability.

We say an instance Π^i_U *accepts* if it holds a session key sk, transcript sid and partner id pid. A client instance can accept at most once. Two instances *terminate* if they accept and do not wish to send any further messages. Usually, PAKE protocols that incorporate confirmation codes distinguish *accept* from *terminate*. In our work in Chapter, J-PAKE and sJ-PAKE protocols do not make this difference, meaning that the instance that accepts also terminates.

Freshness. We define *freshness* to avoid cases where $\mathcal{A}$ might trivially know the bit b, chosen at the beginning of the protocol. Furthermore, we expand the definition of freshness from [ABM15] with more explicit conditions [AB19]. An instance Π^i_U in protocol P is fresh if and only if the following conditions hold:

- the instance was not queried to Test or Reveal before;
- the instance accepted;
- At least one of the conditions hold
  - There exists more than one partner;
  - The instance accepted during a query to Execute;
  - No partner instance exists and Corrupt was not called before it accepted
  - A unique fresh partner instance exists

Forward secrecy. To precisely define forward secrecy, we model two scenarios where established keys are protected:

- Weak forward secrecy guarantees secrecy of session keys by handling scenarios where the adversary is passive and can make a $\mathsf{Corrupt}(U')$ query *any time* and follows up with the $\mathsf{Send}(U, i, m)$ query.
- Perfect forward secrecy protects established session keys for scenarios where the adversary is active, asks a $\mathsf{Corrupt}(U')$ query *before* the Test query, and follows up with the $\mathsf{Send}(U, i, m)$ query.

Advantage of the adversary. We formally define the advantage of the adversary in successfully guessing the bit b, against the protocol P. Let $\mathrm{Succ}_{\mathrm{P}}^{\mathrm{RoR}}(\mathcal{A})$ be the event that $\mathcal{A}$ asks only Test queries to instances Π_U^i that have terminated; at some point $\mathcal{A}$ outputs its guess b' and *wins* if $b' = b$, where b is selected at the beginning of the protocol. The advantage of $\mathcal{A}$ in breaking the security and guessing b is

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{RoR}}() = 2\Pr\left[\mathrm{Succ}_{\mathcal{A}}^{\mathrm{RoR}}()\right] - 1 \tag{4.1}$$

The protocol P is considered to be secure if adversary $\mathcal{A}$ cannot perform any other attack than just online password guessing during an active intervention of the adversary. More formally, we require that for all PPT $\mathcal{A}$:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{RoR}}() \leq \frac{n_{se}}{|D|} + negl(\kappa) \tag{4.2}$$

where n_{se} is the number of Send queries, D is the password dictionary and $negl(\cdot)$ is a negligible function. As already mentioned, we can extend the Equation 4.2 by assuming the password set D has a non-uniform distribution, with some minimal entropy m. This yields

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{RoR}}() \leq \frac{n_{se}}{2^m} + negl(\kappa) \tag{4.3}$$

4.4 Code-based Game-Playing Proofs

Capturing the proof difficulties, reducing errors and easier verification are just some of the reasons Bellare and Rogaway [BR06] created a general framework for game-based proofs in a game-playing setting. They illustrate a powerful tool that makes complex techniques in game-playing proofs easily verifiable and less prone to errors. Furthermore, the game-playing framework translates all the messages (*queries*) between the adversary and the challenger into a code-based game playing, see Figure 4.1. More formally, a game G is a program, viewed as a collection of procedures that are *run* with an *Adversary*, which is also a program, viewed as a single procedure. Each game starts with the procedure *Initialize* that is the first to execute, and it gives input to the procedure Adversary. The Adversary can ask different queries (explained in Section 4.3), called *oracles* by calling out the procedures in the code defined by the game. We consider two

subsequent games identical until a *bad*[4] event happens, and the difference of probabilities of a given outcome is bounded by the probability of that *bad* event happening. Before formalizing this result in the *Fundamental lemma of game-playing* (see Lemma 4.2), we clarify the following notification, used in the same lemma.

Identical-until-bad games. Let K and L be games (programs) and let *bad* be a failure event that occurs in both. Then, K and L are *identical-until-bad* games if their code is the same, but the places where the *bad* occurs is different in both games. We denote $\Pr[\mathcal{A}^K \text{ sets bad}]$ or $\Pr[K^{\mathcal{A}} \text{ sets bad}]$ as the probability that the failure event *bad* is true at the end of the execution of the adversary $\mathcal{A}$ with game K. We denote $Adv(\mathcal{A}^K, \mathcal{A}^L)$ or $Adv(K^{\mathcal{A}}, L^{\mathcal{A}})$ as an advantage that an adversary can obtain in distinguishing a pair of identical-until-*bad* games.

Lemma 4.2. *(Fundamental lemma of game-playing.) Let K and L be identical-until-bad games. Let $\mathcal{A}$ be an adversary. Then*

$$Adv(\mathcal{A}^K, \mathcal{A}^L) \leq \Pr[\mathcal{A}^K \textit{ sets bad}] \textit{ and}$$
$$Adv(K^{\mathcal{A}}, L^{\mathcal{A}}) \leq \Pr[K^{\mathcal{A}} \textit{ sets bad}].$$

When the adversary halts, it possibly gives an output that is passed to the *Finalize* procedure which outputs a string as an outcome of the game. For the generalization and the proof of this lemma, we refer to [BR06]. In our work in Chapter 7, we use the game-playing technique, and for each game and reduction, we provide game-based code[5] in Supplemental material A. Furthermore, we base the indistinguishably of the games on bad events, meaning the games are indistinguishable unless bad events happen.

Proof strategy. The security games are constructed so that values containing the passwords are gradually randomized, i.e., until the the Adversary's view is independent of the passwords used in fresh sessions. We must make sure that:

- The Adversary is allowed to guess only one password per session.
- The values that we randomize do not leak any information about the password.
- At least one of the forms of forward secrecy is preserved.
- We use the reductionist approach to bound the advantage of the Adversary

For more details on game-playing techniques, we refer to [BR06].

[4]We use a flag *bad* as a Boolean variable, setting 1 if a bad event happened and 0 otherwise.
[5]For simplicity, we write the procedures in pseudo-code.

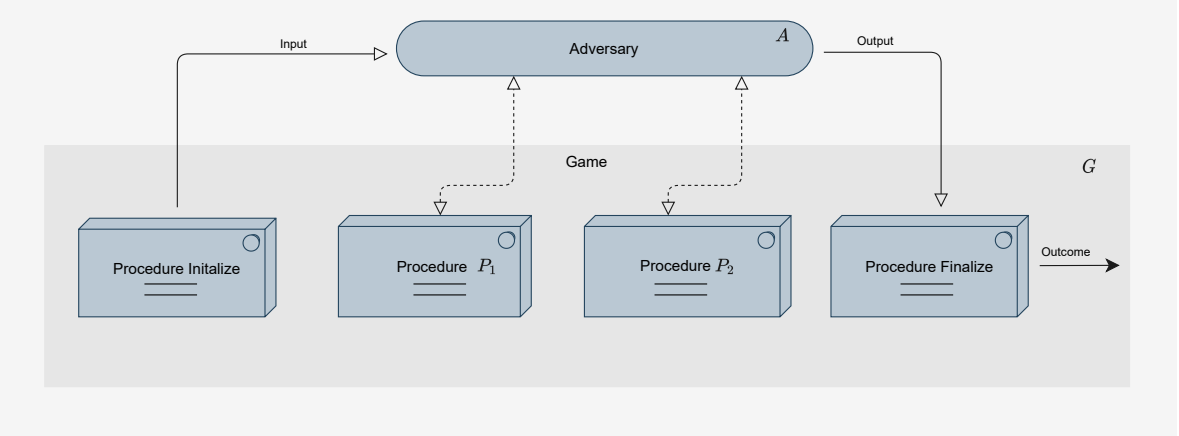

FIGURE 4.1: Running a game G with an adversary A. The adversary is interacting with the game by asking *queries* i.e. calling the oracles that answer its queries.

Nothing in life is to be feared. It is only to be understood.

Maria Curie

5

An Offline Dictionary Attack against zkPAKE Protocol

5.1 Introduction

PAKE allows a user to establish a secure cryptographic key with a server, using only knowledge of a pre-shared password. One of the basic security requirements of PAKE is to prevent offline dictionary attacks. In this chapter, we revisit zkPAKE, an *augmented* PAKE that was proposed by Mochetti, Resende, and Aranha [MRA15]. Our work shows that the zkPAKE protocol is prone to an offline password guessing attack, even in the presence of an adversary that has only eavesdropping capabilities. Furthermore, the results of the performance evaluation show that our attack is practical and efficient. Therefore, we show that zkPAKE is insecure and should not be used as a password-authenticated key exchange mechanism.

5.1.1 Our Contribution

In 2015, Mochetti, Resende and Aranha [MRA15] proposed (without exhibiting a security proof) a simple augmented PAKE called zkPAKE, which they claim is suitable for banking applications, requiring the server to store only the image of a password under a one-way function. Their main idea was to use zero-knowledge proof of knowledge (password) to design an efficient PAKE. However, here we present an offline dictionary attack against the zkPAKE protocol. We also provide a prototype and share the benchmarks

of the attack to demonstrate its feasibility. Our dictionary attack can be carried out in two ways: passively - by eavesdropping on the zkPAKE protocol execution, or actively - by impersonating the server and having the client attempt to log in.

5.1.2 Organization

The rest of the paragraph is organized as follows. First, Section 5.2 describes the zkPAKE protocol and its variant. Then, in Section 5.3, we present an offline dictionary attack against the zkPAKE protocol. Furthermore, in the same section, we offer details about the attack implementation along with the results on the efficiency with different dictionary sizes.

5.2 The zkPAKE Protocol

In this section, we review the zkPAKE protocol. First, we will start with the variant of zkPAKE whose description is presented in Figure 5.1, and then point out the differences with the original design from [MRA15]. The reason for this order of presentation is because the variant of zkPAKE that is proposed later is slightly more elaborate than the original zkPAKE. Hence, we want to show that zkPAKE does not stand against our attack even with proposed modifications.

5.2.1 Protocol Description

zkPAKE, as described in [MRA15] is a two-party augmented PAKE protocol meant to provide authenticated key exchange between a server S and a client C.

5.2.1.1 Initialization Phase

The protocol starts with an enrollment phase, which is executed for every client only once. In this phase, a client and a server (e.g., bank) share a secret value of low entropy that the client can remember. More specifically, in case of zkPAKE, the client must remember the password π, while the server only stores an image of the password R. Before the server computes the corresponding image R; public parameters must be chosen and agreed on: 1) a finite cyclic group $\mathbb{G}$ of prime order q and a random generator g of the group $\mathbb{G}$; 2) Hash functions H_1 and H_2 whose outputs are k-bit strings, where k is the security parameter representing the length of session keys.

5.2.1.2 Protocol Execution

Once the enrollment phase is executed and the public parameters are established, the zkPAKE protocol (see Figure 5.1) will run in three communication rounds as follows:

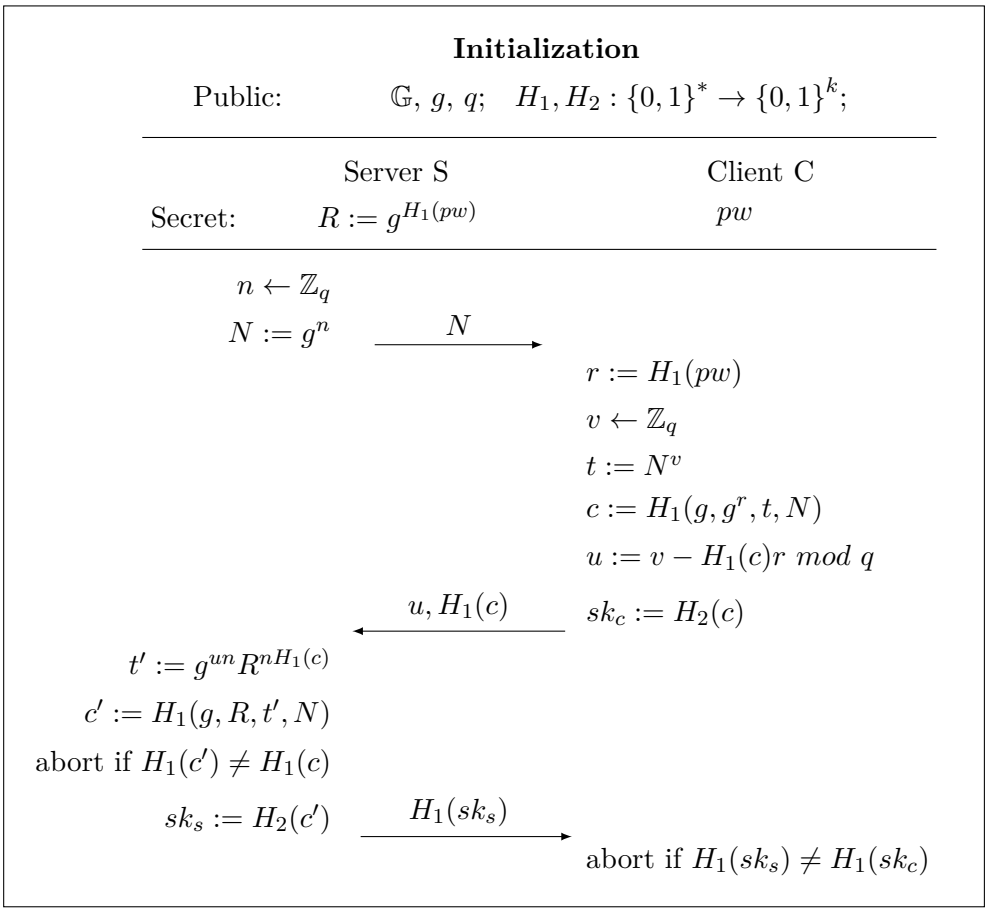

FIGURE 5.1: The zkPAKE protocol.

1. First, the server S chooses a random value n from $\mathbb{Z}_q$, computes N that is supposed to act both as a nonce and Diffie-Hellman value, and sends it to client C.

2. Once receiving the nonce N, client C inputs his password, computes the hash of the password-r, chooses a random element v from $\mathbb{Z}_q$, and computes $t := N^v$. Then, C computes $c := H_1(g, g^r, t, N)$ and obtains $u := v - H_1(c)r$ that should lie in $\mathbb{Z}_q$. Next, C computes the session key $sk_c := H_2(c)$ and sends u and $H_1(c)$ to the S.

3. Upon receiving $H_1(c)$ and u, S recovers t' by computing $g^{un} R^{nH_1(c)}$. Then, S calculates $c' := H_1(g, R, t', N)$. Next, S checks if $H_1(c')$ echoes $H_1(c)$. If it does, S computes the session key $sk_s := H_2(c')$ and sends $H_1(sk_s)$ to C. Otherwise; it aborts the protocol.

4. Similarly, upon receiving $H_1(sk_s)$, C checks if $H_1(sk_s)$ and $H_1(sk_c)$ match. If values are equal, C saves computed session key sk_c and terminates.

As we said before, the authors of zkPAKE have presented two variants of it. The original proposal from [MRA15] differs from the follow-up version in two places: Nonce N is left

underspecified, and value t on the client-side is computed without involving received-nonce. This difference also affects the computation of t' from the server-side. In more detail, the original zkPAKE protocol runs as follows:

1. The server sends his nonce N to client C.

2. The client calculates the hash of his password r, chooses a random parameter $v \leftarrow \mathbb{Z}_q$, and computes $t := g^v$. Then, C computes $c := H_1(g, g^r, t, N)$ and obtains $u := v - H_1(c)r$ in $\mathbb{Z}_q$. Next, C computes the session key $sk_c := H_2(c)$ and sends u and $H_1(c)$ to the S.

3. Upon receiving $H_1(c)$ and u, S recovers t' by computing $g^u R^{H_1(c)}$. Then, S calculates $c' := H_1(g, R, t', N)$. Next, S checks if $H_1(c')$ echoes $H_1(c)$. If it does, S computes the session key $sk_s := H_2(c')$ and sends $H_1(sk_s)$ to C. Otherwise, he aborts the protocol.

4. Finally, upon receiving $H_1(sk_s)$, C checks if $H_1(sk_s)$ echoes $H_1(sk_c)$. If values are equal, C saves computed session key sk_c and terminates.

5.3 Offline Dictionary Attack on zkPAKE

Here, we show that both variants of the zkPAKE protocol are vulnerable to an offline dictionary attack. Our attack exploits the fact that r, which is a hash of the client's password, is of low entropy.

5.3.1 Attack description

Let the enrollment phase be established and let attacker $\mathcal{A}$ be allowed only to eavesdrop on the communication between two honest parties. The attack on the version of zkPAKE protocol presented in Figure 5.1 proceeds as follows:

Step 1. The execution of the protocol starts, and S sends his first message, N. The attacker $\mathcal{A}$ sees the message and stores it in his memory.

Step 2. C does all the computations the protocol demands and sends u and $H_1(c)$ in the second transmission to S. $\mathcal{A}$ observes the second message and obtains u and $H_1(c)$.

Step 3. The adversary now holding N, u and $H_1(c)$ from the first two message rounds may go offline to perform a dictionary attack. His goal is to compute a candidate c' and then use stored $H_1(c)$ as a verifier. The adversary will compute c' by hashing $H_1(g, g^r, t', N)$. Two intermediate inputs to the hash function are obtained by first choosing a candidate password π, and then computing the corresponding r and

t'. Note that the adversary can easily compute $t' = N^v$, since $v := u + H_1(c)r$. Finally, the adversary checks if his guess $H_1(c')$ echoes $H_1(c)$.

Step 4. The adversary repeats Step 3 until he guesses the correct password.

As for the original zkPAKE protocol, the same attack works in a very similar way: Steps 1,2, and 4 are the same, while in Step 3, we need to make a minor change:

Step 3a. The adversary now holding N, u and $H_1(c)$ from the first two message rounds may go offline to perform a dictionary attack. Same as above, the adversary aims to obtain candidate c'_i by computing a hash $H_1(g, g^{r_i}, t'_i, N)$. Here the only difference is that $t'_i = g^{v_i}$, while the formula for computing v_i stays the same.

Note that one can mount a similar dictionary attack by impersonating a server. In this case, the only difference with the eavesdropping attack described above is that the attacker picks the nonce N value. Such knowledge does not additionally help the adversary in our attack. Once the adversary receives the client's reply, he can continue with Steps 3 and 4 from the eavesdropping attack.

5.3.2 Attack Implementation

We implemented a prototype[1] in Python 3 to simulate the attack described above. Our simulation consists of two steps: in the first step, a password is randomly chosen from one of three fixed dictionaries that vary in size. First, the zkPAKE protocol is executed between two honest parties. Then, in the second step (see Algorithm 1), the adversary is given access to honestly generated values as described in Section 5.3.1. The adversary can easily perform an offline dictionary attack against the chosen password with this information in hand.

Algorithm 1: Offline search algorithm

Input: Values N, u, $H(c)$ and a dictionary of passwords P
Output: pw', K
for *each pw' in dictionary* P **do**
  $c' = hash.sha256(pw')$
  $r = c' \bmod q$
  $v = (u + H(c) * r) \bmod q$
  $t = N^v \bmod p$
  $R = g^r \bmod p$
  $H(c') = hash.sha256(g, g^r, t, N)$
  if $H(c) == H(c')$ **then**
    $K = HKDF(c')$
    Return $(pw', H(c'), K)$

[1] Available under GPL v3 at `https://github.com/PetraSala/zkPAKE-attack`.

We performed a set of experiments, using a 224-bit subgroup of a 2048-bit finite field Diffie-Hellman group[2] to determine the time it takes to complete an offline dictionary attack depending on the size of a selected dictionary. Each set of experiments involved mounting the attack by enumerating dictionaries that contain 1000, 10000, and 100000 random password elements. In the end, each experiment was performed 50 times.

5.3.2.1 Results

The times it took the adversarial algorithm described above to find a matching password for each given dictionary are summarized in Table 5.1. Our results demonstrate that

Dictionary size	**Average Time until the Correct Password is found** (ms)	**Std Dev**
1000	3694	1898
10000	27322	17461
100000	244540	178465

TABLE 5.1: Results for dictionary sizes of 1000, 10000, 100000 words.

there is a linear relationship between the size of the dictionary and the average time to find a matching password and shows that an attack is feasible for any adversary with even a small computational power[3]. As expected, the total time for cracking a 100000 password-size dictionary is less than 5 min. Thus, we conclude that the attack would be feasible for dictionaries with significantly more elements. We also note that there are more powerful tools to create more efficient dictionaries, such as HashCat [Has19] or John the Ripper [joh19], which would make the offline search even more effective. The group parameters are taken from the NIST cryptographic toolbox using the 2048 modulus and are shown in Table 5.2.

[2]Selected group parameters, which are originally coming from the standard NIST cryptographic toolbox, are specified in Appendix A.

[3]In all cases the experiments were run under Windows 10 on a 2.8GHz PC with 8GB of memory.

Parameter	Value (Base 16)
Prime Modulus	AD107E1E 9123A9D0 D660FAA7 9559C51F A20D64E5 683B9FD1 B54B1597 B61D0A75 E6FA141D F95A56DB AF9A3C40 7BA1DF15 EB3D688A 309C180E 1DE6B85A 1274A0A6 6D3F8152 AD6AC212 9037C9ED EFDA4DF8 D91E8FEF 55B7394B 7AD5B7D0 B6C12207 C9F98D11 ED34DBF6 C6BA0B2C 8BBC27BE 6A00E0A0 B9C49708 B3BF8A31 70918836 81286130 BC8985DB 1602E714 415D9330 278273C7 DE31EFDC 7310F712 1FD5A074 15987D9A DC0A486D CDF93ACC 44328387 315D75E1 98C641A4 80CD86A1 B9E587E8 BE60E69C C928B2B9 C52172E4 13042E9B 23F10B0E 16E79763 C9B53DCF 4BA80A29 E3FB73C1 6B8E75B9 7EF363E2 FFA31F71 CF9DE538 4E71B81C 0AC4DFFE 0C10E64F
Generator	AC4032EF 4F2D9AE3 9DF30B5C 8FFDAC50 6CDEBE7B 89998CAF 74866A08 CFE4FFE3 A6824A4E 10B9A6F0 DD921F01 A70C4AFA 00C29F52 C57DB17C 620A8652 BE5E9001 A8D66AD7 C1766910 1999024A F4D02727 5AC1348B B8A762D0 521BC98A E2471504 22EA1ED4 09939D54 DA7460CD B5F6C6B2 50717CBE F180EB34 118E98D1 19529A45 D6F83456 6E3025E3 16A330EF BB77A86F 0C1AB15B 051AE3D4 28C8F8AC B70A8137 150B8EEB 10E183ED D19963DD D9E263E4 770589EF 6AA21E7F 5F2FF381 B539CCE3 409D13CD 566AFBB4 8D6C0191 81E1BCFE 94B30269 EDFE72FE 9B6AA4BD 7B5A0F1C 71CFFF4C 19C418E1 F6EC0179 81BC087F 2A7065B3 84B890D3 191F2BFA
Subgroup order	801C0D34 C58D93FE 99717710 1F80535A 4738CEBC BF389A99 B36371EB

TABLE 5.2: The group parameters taken from NIST used for implementation of zkPAKE Protocol.

Mathematics reveals its secrets only to those who approach it with pure love, for its beauty.

Archimedes

6 HoneyPAKEs

6.1 Introduction

This chapter combines two security mechanisms: using a Password-based Authenticated Key Establishment (PAKE) protocol to protect the password for access control and the Honeywords construction of Juels and Rivest to detect loss of password files. The resulting construction combines the properties of both mechanisms: ensuring that the PAKE protocol intrinsically protects the password during transmission and the Honeywords mechanisms for detecting attempts to exploit a compromised password file. Our constructions lead very naturally to two-factor type protocols. An enhanced version of our protocol further provides protection against a compromised login server by ensuring that it does not learn the index to the true password.

6.1.1 Our Contribution

Building on the idea of Juels and Rivest [JR13] we propose a new protocol model called *HoneyPAKE* by merging the design of PAKE with Honeywords, with a goal to add an additional shield for passwords. The proposed protocols are not trying to prevent an attacker from compromising the server and stealing the file of hashed passwords, but to detect such malicious behavior and act accordingly, e.g., raising the silent alarm to the administrator. The alarm raiser would be an additional, secure, simple hardware, *Honeychecker* (HC).

6.1.2 Organization

The rest of this chapter is organized as follows. First, in Section 6.2, we describe the case of access control based on PAKE protocol with an example. Then, in Section 6.3, we give definitions and descriptions of honeywords and the importance of properly generating them and we define a role of a Honeychecker. Then, in Section 6.4, we lay out the security model and discuss possible constructions of HoneyPAKEs. Finally, in Section 6.5, we give an example of how to include authentication of the login server to the client.

6.2 PAKE-based Access control

6.2.1 PPK

A rather elegant protocol, and the one that we will base our construction on, is the PPK protocol due to MacKenzie and Boyko [BMP00], here in simplified form for illustration (H denotes a suitable mapping from the password space to the DH group):

$$\begin{aligned} A \to B: &\quad X := H(s_A) \cdot g^x \\ B \to A: &\quad Y := H(s_B) \cdot g^y \end{aligned}$$

A computes $K_A := (Y/H(s_A))^x$ and B computes $K_B := (X/H(s_B))^y$. These keys match in an honest run if the passwords s_A and s_B match. Online guessing attacks are of course always possible against PAKEs, but observe that here if an active attacker masquerading as one of the parties makes an incorrect guess at the password then the key computed by the legitimate party will be masked by a non-identity term raised to the DH random. This foils offline dictionary attack against terms observed during the protocol and any subsequent key confirmation steps or communications encrypted by the legitimate parties.

6.2.2 PAKE-based Access Control

PAKEs were principally designed to establish secure channels, but the underlying mechanism can be used to protect the password during transmission in an access control protocol. The key confirmation mechanism can be used to authenticate the client to the server. Thus, for example we might adapt PPK to provide authentication of C to S:

$$\begin{aligned} C \to S: &\quad Req_C\,,\, X := H(s_A) \cdot g^x \\ S \to C: &\quad Y := H(s_B) \cdot g^y \\ C \to S: &\quad H_2(K_C) \end{aligned}$$

S confirms that $H_2(K_S) = H_2(K_C)$, where H_2 is a hash function from the group to a compression space. Notice that we inherit the off-line dictionary attack resistance of the PAKE when we base access control on a PAKE. Thus, an attacker masquerading as the login server S will not gain any useful information about the password. This contrasts with a conventional login protocol where the user's (possibly hashed) password, will be revealed to such an attacker.

Remark. In the client-server scenario, the server stores the file F containing password-related information. It is desired that the passwords in F are hashed with a random salt to prevent attacks where the pre-computation of possible passwords immediately discloses the passwords in the clear after the leakage of the file F, e.g. using previously computed rainbow tables. However, since integrating salted passwords with PAKEs is not entirely straightforward, either i) PAKEs do not use salted passwords or ii) the server sends the salt value in clear to the client during the login. Recently Jarecki et al. [JKSS18] proposed a general transformation of PAKE protocols to secure them against pre-computation attacks using an Oblivious PRF. This method could also be applied in our setting.

6.3 Honeywords

Stealing a password file clearly compromises any access control mechanism that uses it. The first step to counter this threat is the well-known idea of storing not the raw passwords but rather crypto hashes of the passwords. Now, when the Access Control (AC) server receives an access request from a user with a password, it computes the hash of the given password and checks that this agrees with the stored hash. Unfortunately, the effectiveness of this counter-measure has diminished as password cracking tools have become more powerful, such as the use of rainbow tables and an increasing number of brute-forcing algorithms. Incorporating salt into the hashes and using slower hash functions helps a bit, but still does not prevent a determined attacker who obtains a password file from extracting the passwords. It thus seems inevitable that password files will be compromised. Ways to distribute shares of the passwords across several remote servers have been proposed in [Boy09, FK00] to compromise such files harder. Still, even this will not guarantee the security of the passwords. Additionally, it would require network infrastructure for password management, and we want to omit such difficulty. The first ones who tackled the problem of password file theft were Bojinov et al. in [BBBB10], where the mention of *honeywords* first appeared. Honeywords were decoys of passwords proposed to set a trap for the attacker who steals a database of passwords to obtain users credentials. The authors in [BBBB10] built a theft-resistant system that generates decoy password sets and forces the attacker to perform many online attempts, which major websites would detect and inhibit. Where Bojinov *et*

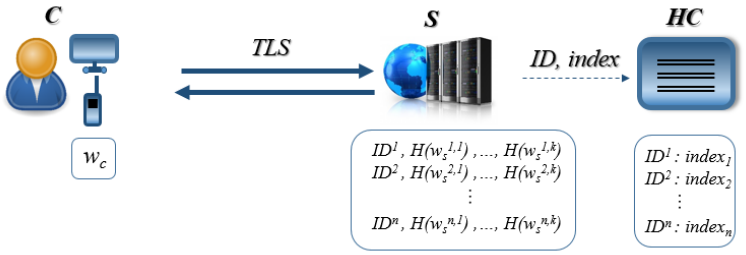

FIGURE 6.1: The Honeywords system of [JR13] is composed of the Honeychecker (HC), the Server (S) and the Client (C).

al. left off, Juels and Rivest continued in [JR13]. They came up with a very simple but effective way to mitigate the effects of password file compromise: not to prevent but rather to detect and perhaps deter exploitation of such a compromise. Instead of storing just the single, correct password, *sugarword*, against the user's identity ID, we store it alongside a number of decoy *honeywords*. Together sugarword and honeywords are called *sweetwords*. The real password will be placed at an arbitrary point in the list and this position is not stored in the file. Logging in is similar to the standard mechanism: Client C provides a putative password, and the Server (S) computes the hash of this, but now it tries to match this against each of the stored hashes. If the proffered password is valid then the server should find a match and it now sends the index of the matching term to the *Honeychecker* (HC). If S finds no match, it will typically notify C that the password is incorrect. The HC should be a separate device linked only to S by a *minimal channel* able to carry only values of type *Index*. HC stores the correct index for each user, and if the index provided by S is correct for the user, it will authorize access. If the index is incorrect then this indicates that, most likely, an attacker is attempting to log in as C using information obtained from a compromised password file. The protocol is thus not fail-safe, but upon intrusion we can let it fail-deadly. Figure 6.1 illustrates the original Honeywords proposal of Juels and Rivest. The proposal of Juels and Rivest requires the following assumptions:

- A *secure channel* between Client and Server prevents an eavesdropper from obtaining the client's password during the authentication phase. In practice, this is typically implemented via a TLS connection. However, it is vulnerable to phishing attacks. In this work, we aim to eliminate this requirement with the help of PAKE-based access control mechanisms.

- *Flatness* on the honey words to ensure that they look plausible alternatives to the real password, i.e. an attacker trying to exploit a stolen password file does not have a better than $1/k$ chance of guessing the true password, where k is the number of sweetwords for that user. We refer to [BBBB10, JR13] for further details about honeywords generation.

FIGURE 6.2: HoneyPAKE system. The Client wants to use the Resource. After running the HoneyPAKE protocol with the login Server S, the Client can access the resource. The credential shared between the Resource and C can be the output of the HoneyPAKE.

6.4 HoneyPAKE

Consider the scenario where C would like to log in to S using his password as means of authentication. We introduce a mechanism that integrates a PAKE protocol into the Honeywords proposal of Juels and Rivest, as shown in Figure 6.2. The resulting system benefits of the security guarantees offered by underlying primitives. More concretely, the idea is i) to detect whenever the password file stored on the Server has been compromised and ii) protect the client's password during its transmission to the S.

6.4.1 The Naive Approach

Incorporating the honeywords idea into PAKEs is not entirely straightforward because S does not know which (hashed) honeyword to use when running the protocol. However, the simplest way to address this is simply to have S not inject any password hash term into the exchanged terms:

$$C \to S: \quad Req_C$$
$$S \to C: \quad Y_S := g^y$$

C now computes $K_C := (Y_S)^x$ and $Z_C := H_2(K_C)$ and sends the following back to S:

$$C \to S: \quad X_C := H(w_C) \cdot g^x \ , \ Z_C$$

Now S computes, for $i \in \{1, \cdots, k\}$

$$W_i := H_2((X_C / H(w_S^i))^y)$$

and compares with Z_C to find the correct hashed password. However, this allows an attacker masquerading as S to launch an off-line dictionary attack: computing W_i for guesses at the password w_S^i until he finds a match. A slightly less naive approach is just running the PPK protocol k times and find a match in one of the runs. This is clearly

rather inefficient, tedious for the user and could leak the index. Therefore, we consider an alternative approach: we introduce a secondary password known to both parties.

6.4.2 Technical Description of Components

We consider a system with three components: the Client, the Server and the Honeychecker which we describe next.

The Client. A legitimate user who would like to connect to server S. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of clients. Each client C_j holds two passwords: a *primary* password and a *secondary* password, which we simply denote by w_C and w'_C and we assume they are chosen uniformly at random from password dictionaries $\mathcal{D}$ and $\mathcal{D}'$ respectively.

The Server is a system in charge of handling clients' login requests. The server S has access to file F storing the clients' passwords. More specifically, the file F stores one entry *per client*, each entry containing the secondary password followed by k *potential* passwords, i.e.:

$$F[C_j] = H(w'_S), H(w^1_S), \ldots, H(w^k_S)$$

where for each, client $C_j \in C$ holding w_C and w'_C as primary and secondary password, it holds that $H(w'_S) = H(w'_C)$ and $\exists\ i$ s.t. $H(w^i_S) = H(w_C)$. The correct index i is not stored by S.

The Honeychecker: This is an auxiliary and simple device whose only goal is to detect whenever the password file F has been compromised. It maintains a list L, by storing the correct index i per client C_j, i.e. $L[C_j] = i$. It accepts two commands:

- Set (C_j, i): Sets $L[C_j]$ to value i.
- Check (C_j, i'): Checks whether $L[C_j]$ equals i'. It outputs a $r = 1$ if $L[C_j] = i'$ and $r = 0$ otherwise.

The connection between S and HC is a *minimal channel* which we assume is secure. The idea is to run a PAKE protocol between C and S to allow S to identify the index i such that $H(w^i_S) = H(w_C)$. Subsequently, S queries the HC with Check(C, i), and the latter will check against its records whether the index i is associated with client C or not. If $r = 1$, it is an indication that a legitimate client is attempting the login, and therefore access to the requested resource should be granted. However, $r = 0$ signals a possible compromise of the password file. In the next section, we describe how a *passive* or *active* HC may react to each scenario.

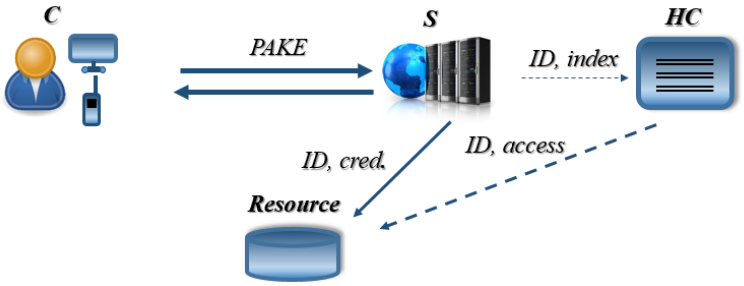

FIGURE 6.3: Login access granted by Resource.

6.4.2.1 Login Access

As described in Figure 6.2 the client wants access to a Resource, e.g., an email service, which may or may not be co-located with the login server. In the HoneyPAKE protocol between the client and the login, the server will, in the end, output a shared key which the server can forward to the resource as a credential for the service (or the established secure PAKE-channel can be used to create a new credential). We do not explicitly write these extra steps in the protocols below, since they may depend on context. The HC can access this login access passively and log the login requests corresponding to correct or wrong indices. The administrator can then periodically check if an alarm was raised, or be alerted immediately. Alternatively, the HC can also play a more active role, see Figure 6.3, and contribute to the decision of whether access is granted or not. The advantage is that malicious attempts to gain access via honeywords will immediately be bounced; however, the downside is the need for a more active HC. The possible cases for login attempts are

- with a correct password, i.e. the sugarword
- with a false password, which is a honeyword
- with a false password, which is not in the honeyword list

The first case will always result in login, while the last possibility will always be blocked by S. The outcome of the second possibility will depend on whether the HC is active or passive.

6.4.3 Security Model

In the security model for the HoneyPAKE system, we will generally consider the HC as being incorruptible. The reasoning behind this assumption is that the HC is a very simple piece of hardware, only handling simple indices. It has minimal external channels; it only needs minimal memory storing indices and handling simple comparisons of indices. On the other hand, the security model does allow the adversary to corrupt S, but only in the form of stealing the password file F. We will discuss stronger forms

of corruption below. One could speculate that extending this model allows the attacker to compromise either S or HC, but not both. Indeed, this will also be secure since the information stored on HC is minimal and would not allow an attacker to compromise security. Indeed, in case HC is compromised, it should not jeopardize the security level of communication between client and server as PAKE protocol protects it. Therefore, in the worst-case scenario, the security level of HoneyPAKEs, even with a corrupted HC, should be at the same security level of any PAKE protocol [BMP00]. We will, however, stay in the model above, which is more closely related to the Honeyword idea and argument of a simple incorruptible HC. Next, we describe the attack scenarios that we consider in this proposal:

1. Compromised File F: As a result of a security breach, adversary $\mathcal{A}$ might get access to the password file F. Regardless of how the passwords are stored in F, e.g. plain text, hashed or hashed and salted, it is reasonable to assume that $\mathcal{A}$ can obtain the passwords in clear by brute-forcing F and then try to masquerade as C to S [BHvOS15].

2. Standard Operation: We consider an adversary who has full control of the communication C between S, different to [JR13], where they assume the existence of a secure channel. However, in this scenario, the attacker does not have access to the password file F.

6.4.3.1 Discussion

Juels and Rivest consider the first attack scenario in [JR13] by introducing the HC as a secondary server. The motivation in [JR13] is not to prevent the leakage of F but to detect whenever such an event occurs. The underlying idea is that whenever F gets compromised, $\mathcal{A}$ may observe at most k *potential* passwords per client, but only one is correct. Furthermore, F contains no information about the index position of the correct password. Then the adversary can only select one candidate password at random when trying to masquerade as C to S. In such a case, the HC could detect the leakage of F with probability $(k-1)/k$ for each attempt of $\mathcal{A}$, and subsequent security measures can be taken e.g., trigger an alarm informing about the compromise of F and asking the server to *reject* the login attempt. We augment the proposal of Juels and Rivest by removing the requirement for a secure channel between C and S. The proposal is to run a PAKE-style protocol between C and S, after which S can identify if C holds one of the k potential hashed passwords $H(w_S^1), \ldots, H(w_S^k)$ and ii) the *potential* index i s.t. $H(w_C) = H(w_S^i)$. Then S proceeds as described in [JR13] by querying the HC which checks if i is the *correct* index or not. The construction guarantees that if the password file F is compromised, an active adversary $\mathcal{A}$ has at most $1/k$ chances of masquerade as C without being detected. In contrast, if F is not compromised, $\mathcal{A}$ can masquerade

as C with success probability at most $1/|D|$, where D is the password dictionary. The second scenario above, called standard operation, is close to the standard PAKE model, and the attacker can be active masquerading as either S or C. However, we do not allow the password file to be compromised in this scenario. The reason is that an adversary knowing the honeyword list of passwords, can actively masquerade as S towards C and do a binary search for the correct password. This would be detectable from the client-side but might not be practical in the real world. We will discuss this below. However, it is reasonable to question why one could not simply store the password file in HC or split it between S and HC and benefit from the assumption that HC is incorruptible. The reason is that the HC is by design an extremely simple component with minimal external channels, as mentioned above. In particular, it is not meant to compute hashes nor to compare or retrieve passwords.

6.4.4 HoneyPAKE Construction

We will now consider our suggestion for a HoneyPAKE protocol. Remember that C holds the two passwords w'_C, w_C and S stores the corresponding password list $H(w'_S), H(w_S^1), \ldots, H(w_S^k)$. The login protocol now runs as follows:

$$\begin{aligned} C \to S: &\quad Req_C \\ S \to C: &\quad Y_S := H(w'_S) \cdot g^y \end{aligned}$$

C now computes $Y = (Y_S / H(w'_C))$ $K_C := Y^x$ and $Z_C := H_2(Y, g^x, K_C)$, $sk = H_3(Y, g^x, g^{xy})$ and sends the following back to S:

$$C \to S: \quad X_C := H(w_C) \cdot g^x \ , \ Z_C$$

Now S computes, for $i \in \{1, \cdots, k\}$

$$W_i := H_2(X, g^y (X_C / H(w_S^i))^y)$$

If $W_i = Z_C$ for some i, then:

$$S \to HC : i$$

and S computes $sk = H_3(Y, g^x, g^{xy})$. HC checks if i agrees with the stored value i^* for C, and if $i \neq i^*$ then an alarm is raised.

6.4.5 HoneyPAKE Security Analysis

In this section, we make a brief and sketchy security analysis. The security of the HoneyPAKE relies on the intractability of the CDH problem in group $\mathbb{G}$. Similar to

other security proofs for PAKE protocols in the random model [BMP00, AP05], to construct a CDH reduction, the confirmation code K_C has to be associated with the identity of the session for which it was computed.[1] We proceed to analyze the security of the HoneyPAKE protocol for *passive* adversaries and sketch a reduction to CDH problem. We give only intuition of the security guarantee for the *active* adversaries and leave the full security proof for future work. We consider the following scenarios:

Scenario 1: Security against eavesdropper adversaries who may have access to password file F.

Claim 1. Honest executions of the protocol between C and S do not leak password information under the CDH assumption.

Proof. Let P_0 be the original protocol. We demonstrate that it is possible to simulate P_0 such that i) no password information is included in the protocol and ii) an eavesdropper $\mathcal{A}_E$ cannot distinguish the original protocol from the simulation except with negligible probability. Let P_1 be such simulation as follows:

$$\begin{aligned} C \rightarrow S: &\quad Req_C \\ S \rightarrow C: &\quad Y_S := g^y \\ C \rightarrow S: &\quad X_C = g^x, Z_C \end{aligned}$$

where $Z_C = H(g^x, g^y, g^z)$ and $x, y, z \xleftarrow{\$} \mathbb{Z}_q$. By inspection, it follows that P_1 does not contain password information. Let E_0 be the event where $\mathcal{A}_E$ queries the random oracle for $H(g^x, g^y, g^{xy})$ such that i) the terms g^x and g^y are generated respectively by C and S in an honest protocol execution. Then obviously, P_0 and P_1 are identical unless the event E_0 occurs, let $Pr[E_0] = \epsilon_0$. We build a CDH-solver $\mathcal{B}^{\mathcal{A}_E}$ whose advantage is ϵ_0/n_{ro}, where n_{ro} is an upper bound to the number of random oracle queries made by $\mathcal{A}_E$. Then it simply follows that P_0 and P_1 are indistinguishable under the CDH assumption. □

Scenario 2: Security against active attackers with no access to password file F. Let $\mathcal{A}$ be an adversary against the HoneyPAKE protocol who fully controls the channel between C and S and does not have corruption capabilities. The construction of the HoneyPAKE intrinsically protects the client's password during the authentication phase, even for hostile networks. It also limits $\mathcal{A}$ to only online dictionary attacks, where she has to guess the primary and secondary passwords for a client of her choice. Let E_2 be the event where $\mathcal{A}$ successfully logs into server S without the HC raising the alarm.

Claim 2. For all adversaries $\mathcal{A}$, $\Pr[E_2] \leq 1/(\mathcal{D} \cdot \mathcal{D}') + \epsilon(\lambda)$, where $\mathcal{D}$ and $\mathcal{D}'$ denote the password dictionaries, ϵ is a negligible function of the security parameter λ.

[1] Typically the session ID is defined as the concatenation of the messages exchanged between C and S without the confirmation code.

Scenario 3: Security against active attackers with access to password file F. In this scenario we allow $\mathcal{A}$ to compromise the server S and obtain the password file F, i.e. for each client, she knows the secondary password w'_C and the list of k potential primary passwords $w_C^1 \cdots w_C^k$. Let E_3 be the event where $\mathcal{A}$ successfully impersonates the client without the HC raising the alarm.

Claim 3. For all adversaries $\mathcal{A}$ with corruption capabilities, $\Pr[E_3] \leq 1/k$.

We do not provide proofs for these claims, but they should follow via standard methods for PAKEs.

Remark. As mentioned above, an adversary, who manages to obtain the password file F and controls the communication between C and S, could try to masquerade as S to C, run the HoneyPAKE protocol and use the client C to obtain the i-th position such that $H(w_C) = H(w_S^i)$. Even though our protocol does not prevent such situations from happening, the client could detect such an attack, which could raise the alarm. Therefore, for our security definition, we assume that an adversary can only compromise the password file but not masquerade as the server.

6.4.6 Variations on a Theme

There are several possibilities for handling the secondary password, that we describe here. We also mention an alternative approach that avoids the need for the secondary password but at a penalty in terms of efficiency. This latter approach does however have some interesting features such as not directly revealing the correct index to S.

Naive Approach: The simplest option is simply to store the hash of the secondary password on the server-side, and either have the user input it each time or store it on the user's device. The former is obviously inconvenient for the user, while the latter makes the protocol device-dependent.

Derived Secondary Password: Rather than storing or re-inputting each time, the secondary password could be computed as a short hash H^* of the $H(w_i)$, where the honeywords for a given user are chosen; they all yield the same short hash value. This, of course, means that there will be a small loss of entropy, a few bits, concerning the already rather low entropy of the usual passwords, but this is probably acceptable.

Secondary Password as Nonce: In place of the secondary password $H(w'_C)$ in the protocol above, we could use a nonce generated by a token for a two-factor type authentication. We assume that each user is provided with a hardware token that will generate short nonces in sync with a similar generator at the server-side, as is done for many internet banking protocols. Such nonces will typically be quite short, low-entropy and easy for the user to type in, so maybe six-digit strings. The purpose of the secondary password, or nonce, is to counter an attacker masquerading as S from launching offline

dictionary attacks. Suppose that such an attacker has managed to guess this value correctly, then this will cancel the value injected by C in computing Z_C. Knowing y, the adversary can now test guesses at the password at leisure by checking for guesses at w_Y:

$$H_2((X_C/H(w_Y))^y) = Z_C$$

Thus, it is enough for the nonce space to be sufficiently large for the chance of guessing correctly to be reasonably small. This is analogous to accepting that there will be a non-negligible chance of a successful online guessing attack against a PAKE. So, the protocol is, as above, with the nonce replacing the hash of the shared password.

6.4.7 HoneyPAKE Without Secondary Password

As remarked earlier, the use of a secondary password may impact usability. Therefore, we can avoid introducing a secondary password, and we discuss some constructions in this section. The setup is as before but without the secondary password.

$$\begin{aligned} C \rightarrow S: &\quad Req_C \\ S \rightarrow C: &\quad X_1 := (H(w_S^1))^y \, , \cdots , \, X_k := (H(w_S^k))^y \end{aligned}$$

C now computes for $i \in \{1, \cdots, k\}$ $Y_i := X_i^x$, and $Y_{k+1} := H_2((H(w_C))^x)$ and sends the following back to S:

$$C \rightarrow S: \quad Y_1 \, , \, Y_2 \, , \cdots , \, Y_{k+1}$$

S now checks if $H_2(Y_i^{1/y}) = Y_{k+1}$ for some i, and if true then:

$$S \rightarrow HC: \quad i$$

This version is less efficient than those presented above and does allow an adversary masquerading as S to have k guessing attempts per faked login. Still, it, avoids the need for a secondary password.

6.4.8 Index-hiding HoneyPAKE

To reduce the scope of online guessing attacks in the last subsection, we can reintroduce the nonce mechanism mentioned above. Further, if C cyclically shifts the terms in the list, we can prevent an honest but curious S from learning the correct password. This addresses a threat scenario that is discussed in [GLRS17]: that of the login server being corrupted and simply recording and later replaying the correct index, perhaps triggered by a cryptic knock. Then, of course, we have to communicate the shift to HC to check if the index is correct. We thus assume that the nonces can be broken into two

concatenated pieces, $Nonce = Nonce_1||Nonce_2$ such that C sees the full string but S sees $Nonce_1$, and HC sees only $Nonce_2$. Thus, $Nonce_1$ protects against online attacks, and $Nonce_2$ disguises the index, and both can be low entropy as above.

$$C \rightarrow S : Req_C$$
$$S \rightarrow C : X_1 := H(Nonce_1) \cdot (H(w_S^1))^y, \cdots, X_k := H(Nonce_1) \cdot (H(w_S^k))^y$$

C now computes for $i \in \{1, \cdots, k\}$ $Y_i := (X_i / H(Nonce_1))^x$, and $Z_c := H_2((H(w_C))^x)$, and cyclically shifts the indices:

$$Z_i := Y_{i+Nonce_2 \ (mod\ k)}$$

and sends the following back to S:

$$C \rightarrow S : Z_1\,,\ Z_2\,,\ ...\,,\ Z_k\,,\ Z_c$$

Now S checks if, for some $j \in \{1, \cdots, k\}$

$$H_2(Z_j^{1/y}) := Z_c$$

If so, then:

$$S \rightarrow HC : j$$

Finally, HC will remove the $Nonce_2$ shift: $j' := j - Nonce_2 \ (mod\ k)$ and check if j' agrees with the stored index. Note that this does not prevent an active adversary who controls S to learn the correct password by replacing passwords in the honeyword list, and check if login is still possible; however, we could make this statistically detectable and auditable by adding an extra round of confirmation codes to be checked by C. An advantage of this protocol over the one in Section 6.4.4, is that an adversary guessing or knowing $Nonce_1$ cannot launch an offline dictionary attack against the password. It follows that if a client accidentally types a password for another service, a malicious S cannot derive this password. A drawback of the protocol in this and the previous subsection is that a malicious client can purposely trigger the honeychecker alarm by changing the order of the returned terms. This could be countered in more advanced but less efficient versions of the protocol. The security of these protocols is based on the CDH or DDH assumption, depending on the type of attack to be prevented. The proofs need a subtly different model than standard PAKE due to the use of secondary passwords. Session Ids and Ids, in general, have been omitted above, but can easily be added for the security proofs.

6.5 Authentication of the Server

In the above, we have focused on authentication of C to S, as befits an access control mechanism. However, it seems wise in certain situations to also authenticate S to C. Our protocols with ephemeral nonces are readily transformable to versions in which S is authenticated to C first, allowing C to abort early if authentication fails. To achieve this C supplies a masked DH term along with the initial request. S can now compute a confirmation code derived from the putative session key which is transmitted back to C in the second message. To illustrate, let us consider a transform of the previous protocol where S also authenticates to C via the shared nonce. The round efficiency is preserved by appending new cryptographic data to the first message which previously only contained the login request:

$$C \rightarrow S : Req_C \, , \, V := H(Nonce_1) \cdot g^z$$

S calculates the confirmation term $X_{-1} := H_2((V/H(Nonce_1))^y)$ and sends it back along with

$$S \rightarrow C : X_{-1} \, , \, X_0 := H(Nonce_1 + 1)g^y \, , \, X_1 := H(Nonce_1 + 1) \cdot (H(w_S^1))^y,$$
$$\cdots \, , X_k := H(Nonce_1 + 1) \cdot (H(w_S^k))^y$$

C now confirms that $X_{-1} = H_2((X_0/H(Nonce_1 + 1))^z)$ and then proceeds exactly as before:

$$C \rightarrow S : Z_1 \, , \, Z_2 \, , \, ... \, , \, Z_k \, , \, Z_c$$

with Z_i as above except 1 is added to $Nonce_i$. And finally S can check whether $H_2(Z_j^{1/y}) := Z_c$ for some $j \in \{1, \cdots, k\}$.

Mathematics knows no races or geographic boundaries: for mathematics, the cultural world is one country.

David Hilbert

7

Security Characterization of J-PAKE and its Variants

7.1 Introduction

The J-PAKE protocol is a Password Authenticated Key Establishment protocol whose security rests on Diffie-Hellman key establishment and Non-Interactive Zero-Knowledge proofs. It has seen widespread deployment and has previously been proven secure, including forward secrecy, in a game-based model. Here, we investigate the extension of such proofs to a significantly more efficient variant of the original J-PAKE, which drops the second round of Non-Interactive Zero-Knowledge proofs, that we call sJ-PAKE. Adapting the proofs to this lightweight variant proves highly-non trivial and requires novel proof strategies in the algebraic group model (AGM). This means that J-PAKE implementations can be made more efficient by simply deleting parts of the code while retaining security under stronger assumptions. We also investigate the security of two further new variants that combine the efficiency gains of dropping the second round NIZK proofs with the gains achieved by two earlier, lightweight variants: RO-J-PAKE and CRS-J-PAKE. The earlier variants replaced the second Diffie-Hellman terms from each party by either a hash term or a CRS term, thus removing the need for half of the NIZK proofs in the first round. We conclude our work by comparing all the variants of J-PAKE and give a brief analysis of efficiency.

7.1.1 Our Contribution

Our starting point is the optimization of J-PAKE shown in Figure 7.3, which we call sJ-PAKE. Compared to the original design, sJ-PAKE drops the zero-knowledge proofs for α and β in the original protocol. In addition, we add a hashing step to derive the shared key. We show that further simplifications that naively omit the zero-knowledge proofs in the first round of the protocol result in insecure protocols. Furthermore, adding an explicit hashing step is also necessary, as the protocol is vulnerable to a related key attack without it.

We then consider the question of formally proving that sJ-PAKE is secure. Our results show that the proposed optimization leads to a surprisingly hard to prove protocol and requires completely different techniques than those used in the original proof of J-PAKE.

Furthermore, we could not find a close adaptation of the original proof of J-PAKE that applies to sJ-PAKE. Nevertheless, we give proof that sJ-PAKE achieves game-based security and provides perfect forward secrecy in the random oracle model, using proof techniques similar to those used for SPAKE2 [AB19]. However, in contrast to SPAKE2, the proof must be carried out in the algebraic group model, even in the case of weak forward secrecy. Intuitively, the reason for this is that SPAKE2 fixes a common reference string that is totally out of the attacker's control in which it is possible to embed a hard problem instance. In contrast, in sJ-PAKE, the adversary has some additional power in choosing the group elements that are used to compute the secret key.

7.1.2 Organization of the Chapter

The rest of the chapter is organized as follows. In Section 7.3 we describe the evolution from J-PAKE to sJ-PAKE. Further, we describe the variants of sJ-PAKE and give a full security game-based proof of sRO-J-PAKE and sCRS-J-PAKE proof in Section 7.6. Finally, in Section 7.7, we compare J-PAKE and sJ-PAKE to all their variants and give a brief conclusion. In addition, we supply game-based code of sJ-PAKE in Supplementary material in A.1.

7.2 J-PAKE and its Variants

For better understanding and clarity of J-PAKE's variants, we first give a brief description of J-PAKE, which is shown in Figure 7.1.

The J-PAKE description. The protocol is symmetric and consists of two rounds. In the first round, a Client chooses two random values x_1 and x_2 from $\mathbb{Z}_q$ and computes $X_1 = g^{x_1}$ and $X_2 = g^{x_2}$. Then it generates proof of knowledge for π_1 for X_1 and π_2 for X_2. A Server does the same, only with x_3 and x_4, computing $X_3 = g^{x_3}$ and $X_4 = g^{x_4}$

and generating π_3 and π_4 for X_3 and X_4, respectively. Then, both sides exchange a tuple of *Identity, Values, Corresponding proofs*, without any order who goes first. Then, both sides verify the proofs π_1, π_2, π_3, π_4 and abort if verification fails.

In the second round, the Client computes $\alpha = g^{(x_1+x_3+x_4)x_2\mathsf{pw}}$, along with corresponding proof π_α for exponent $x_2\mathsf{pw}$, and sends α and π_α to the Server. The Server does the same: computes $\beta = g^{(x_1+x_2+x_3)x_4\mathsf{pw}}$, generates corresponding proof π_β for exponent $x_4\mathsf{pw}$ and sends β and π_β to the Client. Both sides verify the proofs π_α, π_β and abort if verification fails.

In the last step, the Client computes $\mathsf{Key} = (\beta X_4^{-x_2\mathsf{pw}})^{x_2}$ and the Server computes $\mathsf{Key} = (\alpha X_2^{-x_4\mathsf{pw}})^{x_4}$, which results in both sides holding the same key $\mathsf{Key} = g^{(x_1+x_3)x_2x_4\mathsf{pw}}$. The session key follows as $K = H_1(\mathsf{Key})$, where H_1 is a hash function mapping into $\{0,1\}^\kappa$, and κ is the security parameter.

Initialization

Public information: $\mathbb{G}$, g, q, $\sigma \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$; $H_1 : \{0,1\}^* \to \{0,1\}^\kappa$

Secret information: $\mathsf{pw} \in \mathbb{Z}_p, \mathsf{pw} \neq 0$

Client C		Server S
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$		$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_p$
$X_1 \leftarrow g^{x_1}$		$X_3 \leftarrow g^{x_3}$
$X_2 \leftarrow g^{x_2}$		$X_4 \leftarrow g^{x_4}$
$\pi_1 \leftarrow \mathsf{Prv}((X_1, g), x_1, C)$		$\pi_3 \leftarrow \mathsf{Prv}((X_3, g), x_3, S)$
$\pi_2 \leftarrow \mathsf{Prv}((X_2, g), x_2, C)$		$\pi_4 \leftarrow \mathsf{Prv}((X_4, g), x_4, S)$
	$\xrightarrow{(C, X_1, X_2, \pi_1, \pi_2)}$	
	$\xleftarrow{(S, X_3, X_4, \pi_3, \pi_4)}$	
Abort if $X_4 = 1$		Abort if $X_2 = 1$
Check $\mathsf{Ver}((X_3, g), \pi_3, S)$		Check $\mathsf{Ver}((X_1, g), \pi_1, C)$
Check $\mathsf{Ver}((X_4, g), \pi_4, S)$		Check $\mathsf{Ver}((X_2, g), \pi_2, C)$
$\alpha \leftarrow (X_1X_3X_4)^{x_2\mathsf{pw}}$		$\beta \leftarrow (X_1X_2X_3)^{x_4\mathsf{pw}}$
$\pi_\alpha \leftarrow \mathsf{Prv}((\alpha, X_1, X_3, X_4), x_2\mathsf{pw}, C)$		$\pi_\beta \leftarrow \mathsf{Prv}((\beta, X_1, X_2, X_3), x_4\mathsf{pw}, S)$
	$\xrightarrow{\alpha, \pi_\alpha}$	
	$\xleftarrow{\beta, \pi_\beta}$	
Check $\mathsf{Ver}((\beta, X_1X_2X_3), \pi_\beta, S)$		Check $\mathsf{Ver}((\alpha, X_1X_3X_4), \pi_\alpha, C)$
$\mathsf{Key} \leftarrow (\beta X_4^{-x_2\mathsf{pw}})^{x_2}$		$\mathsf{Key} \leftarrow (\alpha X_2^{-x_4\mathsf{pw}})^{x_4}$
$K \leftarrow H_1(\mathsf{Key})$		$K \leftarrow H_1(\mathsf{Key})$

FIGURE 7.1: The J-PAKE protocol

7.3 From J-PAKE to sJ-PAKE

The initial idea of our sJ-PAKE protocol was to slightly modify the J-PAKE protocol by omitting NIZK proofs in the second round and prove the sJ-PAKE secure in the same fashion as Abdalla et al. in [ABM15]. Note that removing the first round NIZK proofs leads to simple offline dictionary attacks in which the receiver incorporates X_1^{-1} in X_3 or X_4 resulting in X_1 cancelling out in the computation of α. However, omitting NIZKs in the second round and computing $K = \mathsf{Key}$ as in J-PAKE leads to so-called *Related key attack* [AP05], where the adversary acts as a *man-in-the-middle* and succeeds in inducing the parties to hold keys with a known relationship. This scenario is not desired, as an adversary can easily create different sessions with related key values. In Figure 7.2, we show the attack, which we describe below.

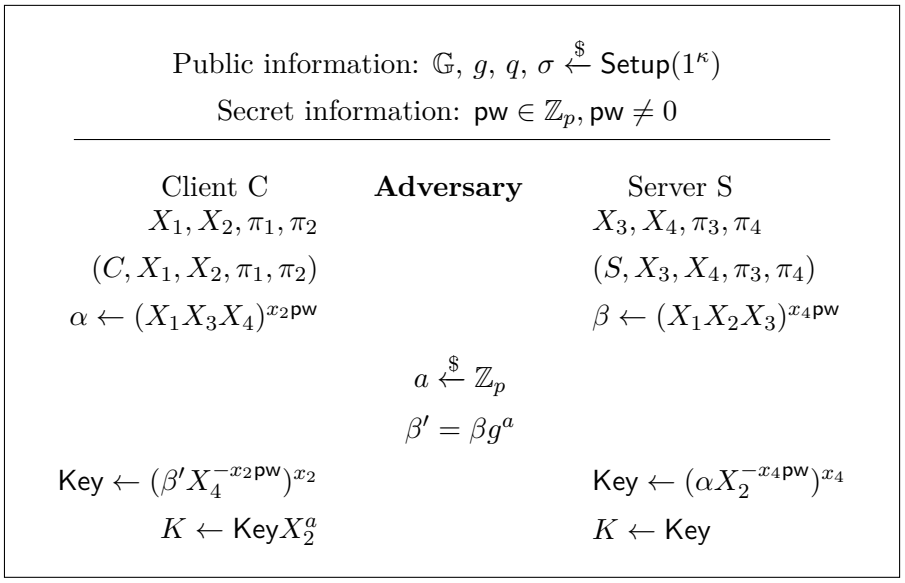

FIGURE 7.2: Related key attack if $K = \mathsf{Key}$.

The first round goes in the same manner as in J-PAKE, with an adversary just forwarding the messages. In the second round, π_C^i computes α and sends it to π_S^j who also computes β and sends it to π_C^i. Then, because there is no proof of knowledge for β, the adversary steps in: intercepts β from π_S^j and computes its own β' as $\beta' = \beta g^a$, for some $a \xleftarrow{\$} \mathbb{Z}_p$ and sends it to π_C^i. Then, π_C^i receives β', computes Key as usual and gets $K_C = \mathsf{Key} X_2^a$, while π_S^j computes its session key and gets $K_S = \mathsf{Key}$. Notice that session keys are linked, and if the adversary tests one instance, it could obtain its session key K and all the adversary needs to do is to compute the other party's session key as $K_S = \frac{K_C}{X_2^a}$. Furthermore, that attack works also for the other party (with resp. to α). Therefore, to prove sJ-PAKE (shown in Figure 7.3) secure, we add the hashing step (the sid, Key and pw included) to derive the session key, which is in any case good practice.

Initialization

Public information: $\mathbb{G}$, g, q, $\sigma \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$; $H_1 : \{0,1\}^* \to \{0,1\}^\kappa$

Secret information: $\mathsf{pw} \in \mathbb{Z}_q, \mathsf{pw} \neq 0$

Client C		Server S
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$		$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$
$X_1 \leftarrow g^{x_1}$		$X_3 \leftarrow g^{x_3}$
$X_2 \leftarrow g^{x_2}$		$X_4 \leftarrow g^{x_4}$
$\pi_1 \leftarrow \mathsf{Prv}((X_1, g), x_1, C)$		$\pi_3 \leftarrow \mathsf{Prv}((X_3, g), x_3, S)$
$\pi_2 \leftarrow \mathsf{Prv}((X_2, g), x_2, C)$		$\pi_4 \leftarrow \mathsf{Prv}((X_4, g), x_4, S)$
	$\xrightarrow{(C, X_1, X_2, \pi_1, \pi_2)}$	
	$\xleftarrow{(S, X_3, X_4, \pi_3, \pi_4)}$	
Abort if $X_4 = 1$		Abort if $X_2 = 1$
Check $\mathsf{Ver}((X_3, g), \pi_3, S)$		Check $\mathsf{Ver}((X_1, g), \pi_1, C)$
Check $\mathsf{Ver}((X_4, g), \pi_4, S)$		Check $\mathsf{Ver}((X_2, g), \pi_2, C)$
$\alpha \leftarrow (X_1 X_3 X_4)^{x_2 \mathsf{pw}}$		$\beta \leftarrow (X_1 X_2 X_3)^{x_4 \mathsf{pw}}$
	$\xrightarrow{\alpha}$	
	$\xleftarrow{\beta}$	
$\mathsf{Key} \leftarrow (\beta X_4^{-x_2 \mathsf{pw}})^{x_2}$		$\mathsf{Key} \leftarrow (\alpha X_2^{-x_4 \mathsf{pw}})^{x_4}$
$K \leftarrow H_1(C, S, X_1, X_2,$		$K \leftarrow H_1(C, S, X_1, X_2,$
$X_3, X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		$X_3, X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$

FIGURE 7.3: The sJ-PAKE protocol.

7.3.1 Variations of J-PAKE

After J-PAKE was proven secure, two more variations of J-PAKE, CRS-J-PAKE and RO-J-PAKE, were proposed and proven secure in the same manner as the original [LST16]. The major difference between them is that CRS-J-PAKE and RO-J-PAKE omit one computation in the first round on each side, for a client-side $X_1 = g^{x_1}$ and a server-side $X_3 = g^{x_3}$. This idea stems from the observation that x_1 and x_3 are not in fact required to compute α, β and K. However, we cannot simply drop the x_1 and x_3 as it is essential for the security to have additional terms in the exponents of α, β and K that are unknown to the parties (other than the x_2 unknown to S and x_4 unknown to C). Thus, the X_1 and X_3 are replaced by terms computed as hashes or by a single CSR term. Fortunately, we can derive variations for sJ-PAKE, sCRS-J-PAKE and sRO-J-PAKE in the same manner. We describe these variations further in the text, explain the differences and compare the efficiency with sJ-PAKE in Section 7.7. The tables of sCRS-J-PAKE and sRO-J-PAKE are shown in Figure 7.5 and Figure 7.7. Keeping

NIZK proofs in two rounds made both CRS-J-PAKE and RO-J-PAKE provable in the same fashion as [ABM15]. However, this is not the case with sJ-PAKE as π_α and π_β are omitted in the second round, which makes its proof very different from [ABM15] and [LST16].

7.4 Game-based Security Proof of sJ-PAKE

This section gives a full detailed proof of sJ-PAKE followed by the security reductions. Further, in Theorem 7.1 we define an advantage of the attacker that uses other attackers as a subroutine running against difficult problems, to break the security of sJ-PAKE.

Theorem 7.1. *Let sJ-PAKE be the protocol described in Fig. 7.3. Take an algebraic RoR attacker $\mathcal{A}$, against sJ-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te}, n_{ro} queries to* Send, Execute, Reveal, Corrupt, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that*

$$\begin{aligned}\mathrm{Adv}^{\mathtt{sJ-PAKE}}_{\mathcal{A}}() \leq\ & \frac{2n_{se}}{\mid D \mid} + \frac{(2n_{se}+4n_{ex})^2}{q} \\ & + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\ & + n_{se}n^2_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK, formally defined in Chapter 2, (Definition 2.9).*

Proof. The proof uses a sequence of games $G_0, G_1, \ldots, G_9$. We follow [BPR00] by including initialize and finalize oracles; the adversary is only allowed to call *Initialize* as its first query and *Finalize* as its final query; games compute their final results as a result of the call to *Finalize*.

The main challenge in the proof with respect to [ABM15] is that we cannot extract the discrete logarithms of adversarial α and β, since ZK-PoK proofs are removed for these elements in the simplified protocol. Instead we must use the RO queries to extract relevant information about the adversary's actions and, in particular, to detect when the adversary is successful in a password guessing attack.

The proof strategy is typical of PAKE protocols. We first modify the security game gradually until the view of the adversary is independent of the passwords used in fresh sessions, which are the ones in which it can obtain an advantage. Then bound

the number of password guesses that the attacker can make to win the game: in the random oracle model, this comes down to bounding the number of RO queries that can be consistent with the trace of a fresh session. In the case of sJ-PAKE we show that, unless the adversary solves a hard problem, only one RO query will satisfy this consistency constraint. This means that the total number of useful password guesses is n_{se} and hence that the adversary's guessing advantage is at most $\frac{n_{se}}{N}$. We give an overview of the sequence of games in Figure 7.4.

Remarks. We denote RO as a random oracle and describe it in Chapter 2 (Section 2.4.1). Furthermore, we describe the model and all queries in Chapter 4 (Section 4.3). In addition, we use the game-playing technique explained in Chapter 4 (Section 4.4), and we display the code of all games and adversaries in Appendix A.1.1.

Useful notation. For each client-server pair (C, S), the game uniformly chooses a password sampled from a password space $\mathcal{P}$, denoted pw_{cs}. In addition, we use notation $T[x]$ to denote access to a dictionary/table T at index x.

- Table T consists of random oracle queries $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$, where a query was assigned to a random value from the game. If there was a query that T does not contain, then the game assigns a new random value and records it; otherwise, it returns the value that was previously assigned to that query.

- Table T_s consists of random oracle queries sid, without including the Key and the pw. It is just a new simulation of T where we replace one random value with another. We call T_s for sessions where the adversary is active.

- Table T_e consists of random oracle queries $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$. We use T_e for sessions where the adversary is passive i.e. for $\mathsf{Execute}$ queries.

- List Corr consists of corrupted instances, and whenever a corruption occurs, we add (C, S) to the list. Thus, there are no fresh instances on the list, Corr.

- List Tst consists of all instances for which $\mathcal{A}$ queried Test query. An instance is fresh if it is not on the list Tst.

Matching sessions. We define matching sessions whenever two honest instances, π_C^i and π_S^j, accept with the same session transcript, sid. More precisely, all sessions resulting from $\mathsf{Execute}$ query or matching Send query, where there was no Send query from

$\mathcal{A}$ for that session, are matching sessions. Note, we consider all sessions resulting from Execute query to be fresh.

Instance state. Each ith instance is denoted as π_U^i has a tuple $(e, \mathsf{sid}, K, \mathsf{ac})$ that describes:

- e is a pair of the dlogs of X_l and X_{l+1}, (x_l, x_{l+1}) for $l = 1, 3$.
- sid is a session transcript of the form $(C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$.
- K is the accepted session key.
- ac is a boolean value that indicates whether the instance accepted ($\mathsf{ac} = \mathsf{T}$) or not ($\mathsf{ac} = \mathsf{F}$).

Further in the proof, we will use $\pi_U^i.e$, $\pi_U^i.\mathsf{sid}$, $\pi_U^i.\mathsf{K}$, $\pi_U^i.\mathsf{ac}$ as the individual component of the state.

Freshness. We define a function $\mathsf{Fresh}(P, i)$ which checks if an instance is fresh by checking all the conditions stated in Chapter 3 (Section 4.3). This function is used in all games until Game 3, and we incorporate it in Game-based code in Supplemental material A. Details about freshness are covered in Game 3.

G_0 **Original Protocol**: The original protocol P.

G_1 **Simulate ZK-PoK proof**s: For Send and Execute query use the NIZK simulator to simulate proof of knowledge. bad_1 is set whenever the adversary detects the difference between the simulated and real proofs.

$G_{1.5}$ **Extract discrete logs from adversarial proofs**. For Send queries use the NIZK extractor to extract witnesses x'_1, x'_2, x'_3, x'_4. $\mathsf{bad}_{1.5}$ is set whenever the adversary submits the proof for which extraction fails.

G_2 **Force unique values**: Repetition of any X values selected from the game are not allowed. bad_2 is set whenever values repeat. We bound the advantage of the adversary by the statistical term.

G_3 **Freshness condition**: Having a unique sid makes the freshness condition explicit in the game. Therefore, instead of running function Fresh and checking all conditions from freshness definition, we check for freshness in every Send query, and we add fr in each state of the instance.

G_4 **Randomize session keys for** Execute **queries**: Session keys are randomized by calling T_e for passive adversaries, and bad_4 is set whenever there was a query in which the correct Key has been computed using pw_{cs}. Thus, the probability of bad_4 happening is reduced to the **CTGDH** problem.

$G_{4.5.1}$ **Randomize** α **for** Execute **queries**: α is randomized for all passive adversaries, and the advantage of the adversary is reduced to the **DDH** problem.

$G_{4.5.2}$ **Randomize** β **for** Execute **queries**: β is randomized for all passive adversaries, and the advantage of the adversary is reduced to the **DDH** problem.

G_5 **Randomize session keys for** Send **queries**: Session keys are randomized by calling T_s, for active adversaries and bad_5 is set whenever there was a query in which the correct Key has been computed using pw_{cs}. bad_5 is not bounded.

G_6 **Detect duplicates**: bad_6 is a sub-event of bad_5 and is set whenever there are two queries with the same sid for $\mathsf{pw}_1 \neq \mathsf{pw}_2$. bad_5 is not bounded and bad_6 is reduced to the **CSqDH** problem.

G_7 **Add algebraic representation**: In Send queries with α and β, we add algebraic representation of α and β. bad_5 is not bounded.

G_8 **Randomize** α **and** β for Send queries: Algebraic adversaries are introduced, and a hybrid argument is used, and the game is split into two hops $G_{7-8.m.1}$ and $G_{7-8.m.2}$, for $1 \leq m \leq n_{se}$.

- $G_{7-8.m.1}$: We tackle the bad scenarios of α and β for Corrupt queries. The advantage of the adversary is reduced to the **DSqDH**.

- $G_{7-8.m.1}$: α and β are randomized and the advantage of the adversary to break the **DDH** is zero unless bad_5 occurs.

G_9 **Perfect forward secrecy.** There are new entries in bad_5, where $\mathcal{A}$ asks for a Corrupt query after the instances accept but before it asks a random oracle query. bad_9^1 may occur if there are new entries where $\mathcal{A}$ tests $\mathsf{pw} \neq \mathsf{pw}_{cs}$ in β' (resp. α') and asks a query with the correct key. If there was a query with the correct key and $\mathcal{A}$ tests $\mathsf{pw} = \mathsf{pw}_{cs}$, then bad_9^2 is set. Furthermoer, bad_5 is bounded by the term $\frac{n_{se}}{|D|}$, bad_9^1 is reduced to the **CSqDH** problem, and bad_9^2 is bounded by $\frac{n_{se}}{|D|}$.

FIGURE 7.4: Description of game-hops for sJ-PAKE

Game 0: Original Protocol.
The first game is the original security game instantiated with sJ-PAKE, so we have

$$\mathrm{Adv}_{\mathcal{A}}^{\mathtt{sJ-PAKE}}() = \mid \Pr[G_0 \Rightarrow \mathsf{T}] - \frac{1}{2} \mid$$

Game 1: Simulate ZK-PoK proofs.
For SendInit-C1, SendInit-S1, and Execute queries, we use SE-NIZK to simulate the *proofs of knowledge* π_1, π_2, π_3, π_4 for X_1, X_2, X_3, X_4 .

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{uzk}}()$$

Game 0 and Game 1 are the same unless the adversary distinguishes real from simulated proofs, and it does it with the advantage $\mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{uzk}}$. We show a reduction for this game in Lemma 7.2. We particularly have use of simulated proofs of knowledge in the reductions, where we will not know the witness for a given challenge.
In this game, we additionally create a list List, where we add only values generated from honest instances, X_1 and X_2 from the client-side, and X_3 and X_4 from the server-side. We need List to clearly distinguish between the values coming from honest instances and values coming from $\mathcal{A}$. This is necessary for Game 1.5, where we will run NIZK.Extract only for adversarial values. In Game 2, we will insist on uniqueness only on the X_1, X_2, X_3 and X_4 values from List.

Game 1.5: Extract discrete logs from adversarial proofs.
For Send-C2 and Send-S2 queries, whenever $\mathcal{A}$ outputs a proof π for a generated X, we run $\mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X, \pi, l)$ which extracts a witness x for X from a valid proof of knowledge π, considering label l (C or S), trapdoor td_e and CRS crs. If the extraction fails, we set $\mathsf{bad}_{1.5}$. Thus, Game 1 and Game 1.5 are the same unless $\mathsf{bad}_{1.5}$ occurs.

$$\mid \Pr[G_1 \Rightarrow \mathsf{T}] - \Pr[G_{1.5} \Rightarrow \mathsf{T}] \mid \leq \Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}]$$

The reduction in Lemma 7.3, to justify this hop, guesses a pair (X, π) where the first $\mathsf{bad}_{1.5}$ happens and, when this event occurs it submits (X, π). The probability that it chooses the correct pair where the extraction failed with the advantage $\mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{ext}}$ can be bound:

$$\Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}] \leq 2n_{se}\mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{ext}}()$$

Game 2: Force unique values
Repetition of any values X_1, X_2, X_3 or X_4, previously seen in an execution, is not allowed. We set bad_2 whenever values, X_1, X_2, X_3 or X_4, repeat. Game 2 is the same as Game 1, unless bad_2 occurs. More precisely, we have two values for SendInit-C1 and

SendInit-S1, four values for Execute queries and the size of the group q:

$$\Pr[G_{1.5} \Rightarrow \mathsf{T}] - \Pr[G_2 \Rightarrow \mathsf{T}] \leq \frac{(2n_{se} + 4n_{ex})^2}{q}$$

This means that $2n_{se} + 4n_{ex}$ is a maximal number of values that can repeat, and we can bound the bad event by the birthday paradox bound in case X_1, X_2, X_3 or X_4 collide. In case of a bad event, we mark an instance for which values repeat as "Invalid".

Game 3: Adding freshness.
Now that we have unique sessions, and we add explicit freshness conditions for all sessions. These are initially false and then set to their correct value when the session accepts in Send-C3 and for Send-S3. A fresh session may later become unfresh if its key is given to the adversary as a result of a Reveal query. We also remove all bad events. This change does not affect the view of the adversary, so we have:

$$\Pr[G_3 \Rightarrow \mathsf{T}] = \Pr[G_2 \Rightarrow \mathsf{T}]$$

Game 4: Randomize session keys for matching sessions.
We no longer use the random oracle for matching sessions to derive the key. We use a totally random key instead. We keep these removed entries in a new list T_e. If a random oracle query is placed at any point that could cause an inconsistency in the adversary's view, we set a bad flag bad_4. Note, for Send queries; we check if the sessions are matching in Send-S3 and Send-C3.

$$|\Pr[G_3 \Rightarrow \mathsf{T}] - \Pr[G_4 \Rightarrow \mathsf{T}]| \leq \Pr[G_4 \Rightarrow \mathsf{bad}_4]$$

The reduction to justify this hop guesses the l^{th} Execute query where the first bad_4 happens and, when this event occurs, it solves an instance of the Computational Triple Group problem, CTGDH, so we have:

$$\Pr[G_4 \Rightarrow \mathsf{bad}_4] \leq n_{ro} n_{ex} \mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Game 4.5: Randomize alpha/beta for Execute queries.
We randomize α and β for sessions resulting from Execute queries using two intermediate games where we first randomize α in Game 4.5.1, then β in Game 4.5.2.

Game 4.5.1 Randomize α.
We randomize α in sessions resulting from Execute queries. Game 4.5.1 is indistinguishable from Game 4 unless $\mathcal{A}$ solves the DDH problem. The reduction is shown in Lemma

7.5.

$$|\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}]| \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

Game 4.5.2 Randomize β.
Now, we randomize β in sessions resulting from Execute queries. Game 4.5.2 is indistinguishable from Game 4.5.1 unless $\mathcal{A}$ solves the DDH problem. The reduction is shown in Lemma 7.6.

$$|\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}]| \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Game 5: Randomize session keys for Send queries.
We randomize session keys for all Send queries, i.e. for active attacks. Concretely, we no longer use the random oracle to compute the output key of such sessions when they are fresh and use a totally random key instead. We introduce a table T_s to keep track of the inputs excluded from the random oracle input. In case of a random oracle query that could cause an inconsistency in the adversary's view, we set a bad flag bad_5. Therefore, we naturally have:

$$|\Pr[G_{4.5.2} \Rightarrow \mathsf{T}] - \Pr[G_5 \Rightarrow \mathsf{T}]| \leq \Pr[G_5 \Rightarrow \mathsf{bad}_5]$$

Game 6: Detect duplicates.
Rather than checking for bad_5 in the game we, construct a list of random oracle queries that would cause bad_5 and, at the end of the game, we divide the checking for bad_5 into two sub-events. Firstly, we set a new bad flag if ever there are two entries in the list of problematic random oracle queries that are consistent with the same sid, i.e., if there exist queries $(\mathsf{sid}, \mathsf{Key}_1, \mathsf{pw}_1)$ and $(\mathsf{sid}, \mathsf{Key}_2, \mathsf{pw}_2)$, where Key_1 and Key_2 are computed correctly, for $\mathsf{pw}_1 \neq \mathsf{pw}_2$. Such a pair of queries we call *duplicates* and whenever we detect them, we set bad_6. Secondly, if this event does not occur and the list of problematic queries is not empty, we set bad_5. In this game, by bounding bad_6, we restrict the set of executions where bad_5 may occur, so we have:

$$\Pr[G_5 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_6 \Rightarrow \mathsf{bad}_5] + \Pr[G_6 \Rightarrow \mathsf{bad}_6]$$

The reduction for this game guesses the l^{th} session where the first bad_6 happens, together with the indices of the two problematic random oracle queries, and reduces the bad event to the Computational Square Diffie-Hellman problem, CSqDH. This implies:

$$\Pr[G_6 \Rightarrow bad_6] \leq n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

The reduction is shown in Lemma 7.7.

Game 7: Add algebraic representation.
In this hop, we keep the check for bad_5 as before, and we introduce algebraic adversaries by adding algebraic representation whenever there is Send query with α or β. Therefore, we have:

$$\Pr[G_6 \Rightarrow \mathsf{bad}_5] = \Pr[G_7 \Rightarrow \mathsf{bad}_5]$$

Game 8: Randomizing α and β for Send queries.

$$\Pr[G_7 \Rightarrow \mathsf{bad}_5] - \Pr[G_8 \Rightarrow \mathsf{bad}_5] \leq 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

We use a hybrid argument to randomize α and β for all Send queries, one session at a time. This means, if we consider the m^{th} session, then all sessions before m^{th} will have random α and β, and for all sessions after m^{th}, we will compute α and β normally. However, we cannot randomize messages with α and β in one go, because of the following scenario. Let us say $\mathcal{A}$ communicates with a client instance. In the second round, the Client chooses α randomly and sends it to $\mathcal{A}$. Then $\mathcal{A}$ decides to corrupt and send β' as $\beta' = X_4'^{x_2\mathsf{pw}_{cs}} g^s$, where s is something that only $\mathcal{A}$ knows. Here, we face a problem because $\mathcal{A}$ can fix any s it wants, and our reduction will not be able to detect what $\mathcal{A}$ is doing and respond with the appropriate key. For this reason, we assume we are dealing with algebraic adversaries, meaning, that whenever $\mathcal{A}$ outputs β', it also outputs alg' that gives the representation of β' in terms of other group elements it observed during the game. These group elements are g, and honestly generated X_1, X_2 and α. This analysis means that any group element that $\mathcal{A}$ produces can be rewritten as a representation of the form $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$. We start the change for m^{th} session, for $1 \leq m \leq n_{se}$ and we split Game 8 into two games: Game $7-8.m.1$ where $\mathcal{A}$ sends β' with alg', we compute α honestly and use alg' to decompose β' and Game $7-8.m.2$, where we randomize α.

Game 7-8.m.1 We make the change in all instances before the m^{th} session, and for all instances after mth, we simulate as in Game 7. Let us say, algebraic adversary corrupts the session with an honest client instance right after it receives an honestly computed α, i.e. $\alpha = (X_1 X_3' X_4')^{x_2\mathsf{pw}_{cs}}$, but before sending β'. Then $\mathcal{A}$ sends β' with $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$, and we use $\mathsf{Rewrite}$ that rewrites β' as $\beta' = g^a X_1^b X_2^c \alpha^d$. Then, we reduce the construction of $\mathsf{Key} = (\frac{\beta'}{X_4'^{x_2\mathsf{pw}_{cs}}})^{x_2}$ by checking two cases:

(a) In the case of $\mathsf{Corrupt}$, if $X_2^c \alpha^d \neq X_2^{x_4'\mathsf{pw}_{cs}}$, then $\mathcal{A}$ gets a random key. Furthermore, $\mathcal{A}$ needs to solve a Decisional Square Diffie-Hellman problem, DSqDH to compute the key.
(b) In the case of $\mathsf{Corrupt}$, if $X_2^c \alpha^d = X_2^{x_4'\mathsf{pw}_{cs}}$, then we compute the real key, i.e. $\mathsf{Key} = (\frac{\beta'}{X_4^{x_2\mathsf{pw}_{cs}}})^{x_2}$. Furthermore, in case ($b$), after rearranging the exponents, we get

$\mathsf{Key} = g^{(x_2a+x_1x_2b)} = g^{x_2a}g^{x_2x_1b}$. Now, the final thing we need to do in (b), is to embed α in the Key preparing for the next hop, and we get $\mathsf{Key} = g^{x_2a}(\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_2(x'_3+x'_4)}})^b$. We tackle the keys for fresh sessions as before. We stress that the only change in Game $7-8.m.1$ is in case of corruption in Send-C3 or Send-S3. If that does not happen we simulate everything as in Game 7.

$$|\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}]| \leq 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

We reduce the advantage of the adversary in Game 7-8.m.1 to the DSqDH, shown in Lemma 7.8.

Game 7-8.m.2. For all instances before m^{th}, we make the change, and for all instances after m^{th}, we simulate as in Game 7-8.m.1. In this hop we randomize α.

$$|\Pr[G_{7-8.m.1} \Rightarrow T] - \Pr[G_{7-8.m.2} \Rightarrow T]| \leq \Pr[G_8 \Rightarrow \mathsf{bad}_5]$$

In this game, the test bit b is perfectly hidden in fresh sessions, so the adversary has zero advantage in noticing that α is randomized. Therefore, Game 7-8.m.1. and Game 7-8.m.2. are the same unless bad_5 occurs. We stress that we do the same hybrid analysis for randomizing β.

Game 9: Perfect forward secrecy.

In this hop, it remains to bound bad_5 and address the cases where $\mathcal{A}$ asks for a $\mathsf{Corrupt}$ query after the instances accept but before it asks a random oracle query. Everything is randomized in Send and $\mathsf{Execute}$; therefore, we can delay password sampling until the end or in case of corruption. Now, among new entries that might cause bad_5, there could be entries where $\mathcal{A}$ sends a message with β' (resp. α'), where it tested a different password, $\mathsf{pw}' \neq \mathsf{pw}_{cs}$ and then later asked a query with pw_{cs}. In that case, we set a bad flag bad^1_9. Now, it might also happen that $\mathcal{A}$ sent β' (resp. α') where $\mathsf{pw}' = \mathsf{pw}_{cs}$. In that case, we set a bad flag bad^2_9. More precisely, we split bad_5 after corruption into bad^1_9 and bad^2_9. Let us say, $\mathcal{A}$ sent β' with alg' and then accepted. Then, $\mathcal{A}$ corrupts and asks for a random oracle query $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$. Thanks to the algebraic adversary, we can use alg of β' it produced in terms of g, X_1, X_2 and α, which it observed in the game. These elements are computed honestly, by the game, $X_1 = g^{x_1}$, $X_2 = g^{x_2}$ and $\alpha = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. In other words, β' can be rewritten as a representation of the form $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$. Therefore, we use the function $\mathsf{Rewrite}$ to decompose β' to $\beta' = g^a X_1^b X_2^c \alpha^d$. Then, in case of corruption, we reduce the construction of $\mathsf{Key} = (\frac{\beta'}{X_4'^{x_2\mathsf{pw}_{cs}}})^{x_2}$ by checking two cases:

(a) In the case of $\mathsf{Corrupt}$, if $g^{x_2c} \neq X_4^{x_2\mathsf{pw}_{cs}}$, then $\mathsf{pw}' \neq \mathsf{pw}_{cs}$. And we set bad^1_9.

In this case, $\mathcal{A}$ needs to solve a Computational square Diffie-Hellman problem to compute the correct key.
(b) In the case of $\mathsf{Corrupt}$, if $g^{x_2 c} = X_4^{x_2 \mathsf{pw}_{cs}}$, we set bad_9^2. It means that $\mathsf{pw}' = \mathsf{pw}_{cs}$ and $\mathcal{A}$ submitted the winning query. To compute the key, the expressions cancel, and we get $\mathsf{Key} = g^{(x_2 a + x_1 x_2 b + x_2 w d)} = g^{x_2(a + x_1 b + w d)}$.
We check bad_5 as before. Note that bad_9^1 and bad_9^2 are disjoint. So, we have the following bound

$$\Pr[G_8 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_9 \Rightarrow \mathsf{bad}_5] + \Pr[G_9 \Rightarrow \mathsf{bad}_9^1] + \Pr[G_9 \Rightarrow \mathsf{bad}_9^2]$$

The entries where bad_5 might happen are in the list which size is now at most n_{se}, so we have:

$$\Pr[G_9 \Rightarrow \mathsf{bad}_5] \leq \frac{n_{se}}{\mid D \mid}$$

We reduce the probability of bad_9^1 to Computational Square Diffie-Hellman, and we show the reduction in Lemma 7.9, where we plug in the challenge for the l^{th} session and guess the entry for which bad_9^1 might have occurred. For this reason, we have

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq n_{se} n_{ro} \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()$$

Finally, the number of remaining entries that can cause bad_9^2 is at most $\frac{n_{se}}{|D|}$, so we have

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq \frac{n_{se}}{\mid D \mid}$$

This completes the proof and yields the bound.

$$\begin{aligned}\mathrm{Adv}^{\mathtt{sJ-PAKE}}_{\mathcal{A}}() \leq \quad & \frac{2n_{se}}{\mid D \mid} + \frac{(2n_{se} + 4n_{ex})^2}{q} \\ & + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()\end{aligned}$$

□

7.5 Reductions

Here, we prove auxiliary lemmas supporting the proof of Theorem 7.1.

Lemma 7.2. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ such that*

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathrm{uzk}}_{\mathcal{B}_{\mathsf{NIZK}}}()$$

Proof. Further, we consider $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ attacker in Figure A.3 and we prove that the attacker distinguishes real from random proofs with the advantage $\mathrm{Adv}^{\mathrm{uzk}}_{\mathcal{B}_{\mathsf{NIZK}}}$. We simulate proofs in the following queries. For SendInit-C1(C, i, S), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_1 = g^{x_1}$ and $X_2 = g^{x_2}$ and calls simulation oracle $\mathsf{Sim_O}$ that takes X_1 and C and outputs π_1. Then $\mathsf{Sim_O}$ takes X_2 and C and outputs π_2. For SendInit-S1(S, i, C), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_3 = g^{x_3}$ and $X_4 = g^{x_4}$ and calls $\mathsf{Sim_O}$ that takes X_3 and S and outputs π_3. Then $\mathsf{Sim_O}$ takes X_4 and S and outputs π_4. For Execute(C, S, i, j), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_1 = g^{x_1}$, $X_2 = g^{x_2}$, X_3 and π_4 for X_4 and calls $\mathsf{Sim_O}$ to simulate π_1, π_2, π_3, π_4 for the corresponding X_1, X_2, X_3, and X_4. The attacker interpolates between Game 0 and Game 1, meaning if it knows it is playing Game 0, it outputs 1, and if it knows it is playing Game 1, it outputs 0. □

Lemma 7.3. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ such that*

$$\Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}] \leq 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}()$$

Proof. We consider $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ attacker in Figure A.5. We set a bad flag $\mathsf{bad}_{1.5}$ whenever the extractor fails, and we denote the advantage of the attacker as $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}$. Further, we check if the extractor returns the correct witness in the following queries. In Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$, we check if the received value X'_3 comes from the adversary (not in List) in which case we call $\mathsf{NIZK.Extract}(\mathsf{crt}, \mathsf{td}_e, X'_3, \pi'_3, S)$ which outputs a witness x'_3. Now, we check if $X'_3 = g^{x'_3}$, in which case the witness is valid. Otherwise, either the extraction fails and $\mathsf{bad}_{1.5}$ occurred, or the witness is not valid. In either case we add the pair (X'_3, π'_3) to a list $\mathsf{List}_{\mathsf{Ext}}$. We do the same for X'_4 and in Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$ for X'_1 and X'_2. At the end of the game, $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ guesses one pair (X', π') from $\mathsf{List}_{\mathsf{Ext}}$ for which $\mathsf{bad}_{1.5}$ might have occurred. Thus, we can bound the probability of $\mathsf{bad}_{1.5}$ happening with $2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}$. □

Lemma 7.4. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_4$ such that*

$$\Pr[G_4 \Rightarrow \mathsf{bad}_4] \leq n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Proof. We consider a CTGDH attacker $\mathcal{B}_4$ in Figure A.9. The attacker $\mathcal{B}_4$ gets a challenge of the form $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$ and finds $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$. $\mathcal{B}_4$ uses the challenge for the ℓ^{th} $\mathsf{Execute}$ query to compute $X_1 = g^{x_1}$, $X_2 = X$, $X_3 = Y$, $X_4 = Z$, $\alpha = (D_{XY}D_{XZ}X^{x_1})^{\mathsf{pw}_{cs}}$ and $\beta = (D_{XZ}D_{YZ}Z^{x_1})^{\mathsf{pw}_{cs}}$. If there is a query with a collision in T, we add it to the list T_4. Then, when the game finishes, the reduction guesses one query from T_4 for which bad_4 might have occurred. If the guess was successful, it means that bad_4 occurred and $\mathcal{A}$ solved $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$ so $\mathcal{B}_4$ can recover g^{xyz} by computing:

$$g^{xyz} = \frac{\mathsf{Key}^{\frac{1}{\mathsf{pw}_{cs}}}}{D_{XZ}^{x_1}}$$

For Send queries, the attacker runs everything as in Game 4. □

Lemma 7.5. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{4.5.1}$ such that*

$$\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

Proof. We consider a DDH attacker $\mathcal{B}_{4.5.1}$ in Figure A.11. The attacker $\mathcal{B}_{4.5.1}$ gets a generalized challenge of the form $(X_1, ..., X_{n_{se}}, Y_1, ..., Y_{n_{se}}, Z_1, ..., Z_{n_{se}})$ and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge for each Execute query to compute $X_1 = X_k$, $X_2 = Y_k$, $X_3 = g^{x_3}$, $X_4 = g^{x_4}$, $\alpha = (Z_k X_2^{x_3+x_4})^{\mathsf{pw}_{cs}}$ and $\beta = X_2^{x_4 \mathsf{pw}_{cs}} g^{(x_1+x_2)x_4 \mathsf{pw}_{cs}}$. Further, $\mathcal{B}_{4.5.1}$ interpolates between Game 4 and Game 4.5.1, meaning if (X_k, Y_k, Z_k) is a real DDH tuple, it is playing Game 4, and otherwise, it is playing Game 4.5.1. □

Lemma 7.6. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{4.5.2}$ such that*

$$\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Proof. We consider a DDH attacker $\mathcal{B}_{4.5.2}$ in Figure A.13. The attacker $\mathcal{B}_{4.5.2}$ gets a generalized challenge of the form $(X_1, ..., X_{n_{se}}, Y_1, ..., Y_{n_{se}}, Z_1, ..., Z_{n_{se}})$ and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge for each Execute query to compute $X_1 = g^{x_1}$, $X_2 = g^{x_2}$, $X_3 = X_k$, $X_4 = Z_k$, $\alpha \xleftarrow{\$} \mathcal{G}$ and $\beta = (Z_k X_4^{x_1+x_2})^{\mathsf{pw}_{cs}}$. Further, $\mathcal{B}_{4.5.2}$ interpolates between Game 4.5.1 and Game 4.5.2, meaning, if (X_k, Y_k, Z_k) is a real DDH tuple, it is playing Game 4.5.1, and otherwise, it is playing Game 4.5.2. □

Lemma 7.7. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_6$ such that*

$$\Pr[G_6 \Rightarrow \mathsf{bad}_6] \leq n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

Proof. We consider a CSqDH attacker $\mathcal{B}_6$ in Figure A.16. The attacker $\mathcal{B}_6$ gets a challenge X and finds the solution to CSqDH. The attacker $\mathcal{B}_6$ guesses the ℓ^{th} fresh session, where it uses the challenge for SendInit-C1(C, i, S) query to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$ to compute $\alpha = X_2^{(x_1+x_3'+x_4')\mathsf{pw}_{cs}}$. We add all queries where bad_6 might have occurred to T_{bad}. At the end of the game, $\mathcal{B}_6$ guesses one query from T_{bad}. If the guess was successful, it means $\mathcal{A}$ solved CSqDH and

$\mathcal{B}_6$ can recover g^{x^2} in the following way:

$$\mathsf{Key}_1 = (\frac{\beta'}{X^{x'_4\mathsf{pw}_{cs}}})^x = (\frac{g^{(x_1+x'_3)x'_4\mathsf{pw}_1}X^{x'_4\mathsf{pw}_1}}{X^{x'_4\mathsf{pw}_{cs}}})^x = X^{(x_1+x'_3)x'_4\mathsf{pw}_1}g^{x^2x'_4(\mathsf{pw}_1-\mathsf{pw}_{cs})} \quad (1)$$

$$\mathsf{Key}_2 = (\frac{\beta'}{X^{x'_4\mathsf{pw}_{cs}}})^x = (\frac{g^{(x_1+x'_3)x'_4\mathsf{pw}_2}X^{x'_4\mathsf{pw}_2}}{X^{x'_4\mathsf{pw}_{cs}}})^{x_2} = X^{(x_1+x'_3)x'_4\mathsf{pw}_2}g^{x^2x'_4(\mathsf{pw}_2-\mathsf{pw}_{cs})} \quad (2)$$

where x is unknown. From (1) and (2) follows:

$$g^{x^2} = (\frac{\mathsf{Key}_1}{\mathsf{Key}_2X^{(x_1+x'_3)x'_4(\mathsf{pw}_1-\mathsf{pw}_2)}})^{\frac{1}{x'_4(\mathsf{pw}_1-\mathsf{pw}_2)}}$$

We do the same procedure on the server-side and plug-in the challenge for SendInit-S1(S,i,C). Note, in the case of a $\mathsf{Corrupt}$ query in ℓ^{th} session, the session is no longer fresh, and bad_6 cannot occur. Furthermore, it means that the attacker made the wrong guess, so we abort. □

Lemma 7.8. *For every algebraic attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{7-8.1}$ such that*

$$\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}] \leq 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

Proof. We consider a DSqDH attacker $\mathcal{B}_{7-8.1}$ in Figure A.19. The attacker $\mathcal{B}_{7-8.1}$ gets a challenge (X,Y) and submits $b=1$ if (X,Y) is a real DSqDH tuple or $b=0$ otherwise. The attacker uses a hybrid argument and for the fresh m^{th} session, to plugs-in the challenge for SendInit-C1(C,i,S) to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C,i,S,X'_3,X'_4,\pi'_3,\pi'_4)$ to compute $\alpha = X_2^{(x_1+x'_3+x'_4)\mathsf{pw}_{cs}}$. In Send-C3$(C,i,S,\beta',\mathsf{alg}')$ where $\mathsf{alg}' = [((g,a),(X_1,b),$
$(X_2,c)),(\alpha,d)]$, in case of $\mathsf{Corrupt}$, we rewrite β' as $\beta' = g^aX_1^bX_2^c\alpha^d$. Then we check:

(a) If $X_2^c\alpha^d \neq X_2^{x'_4\mathsf{pw}_{cs}}$, we compute $\mathsf{Key} = X_2^a(\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{X_2^{(x'_3+x'_4)}})^bY^{(c+d(x_1+x'_3+x'_4)\mathsf{pw}_{cs}-x'_4\mathsf{pw}_{cs})}$. This means if $Y = g^{x^2}$ then the key is real and random otherwise.

(b) If $X_2^c\alpha^d = X_2^{x'_4\mathsf{pw}_{cs}}$, we embed α in the key and we compute it as $\mathsf{Key} = X_2^{(a+x_1b)} = X_2^a(\frac{\alpha^{\frac{1}{\mathsf{pw}}}}{X_2^{(x'_3+x'_4)}})^b$.

We say $\mathcal{A}$ interpolates between the two games, meaning, if $Y = g^{x^2}$ then it is playing Game 7, and otherwise, it is playing Game 7-8.m.1. Note, we do the same analysis on the server's side, for SendInit-S1(S,i,C), Send-S2$(S,i,C,X'_1,X'_2,\pi'_1,\pi'_2)$ and Send-S3$(S,i,C,\alpha',\mathsf{alg}')$, to tackle with problematic cases of α' (coming from $\mathcal{A}$) . □

Lemma 7.9. *For every algebraic attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_9$ such that*

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()$$

Proof. We consider a CSqDH attacker $\mathcal{B}_9$ in Figure A.22. The attacker $\mathcal{B}_9$ gets a challenge X and finds the solution to the CSqDH. The attacker guesses the ℓ^{th} fresh session, where instances accept, and there was a Corrupt query, and a query $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$ where there was β' (resp. α') with $\mathsf{pw} \neq \mathsf{pw}_{cs}$. Further, $\mathcal{B}_9$ plugs-in the challenge for SendInit-C1(C, i, S) query to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$ to compute $\alpha = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. We add all queries where bad_9^1 might have occurred to a list T_9. At the end of the game, the attacker guesses one query from T_9. If the guess was successful, it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_9$ can recover g^{x^2}:

$$g^{x^2} = \left(\frac{\mathsf{Key}}{X^{(a+x_1 b+wd)}}\right)^{\frac{1}{c-x_4 \mathsf{pw}_{cs}}}$$

We do the same procedure on the server-side and plug-in the challenge in SendInit-S1(S, i, C). □

7.6 Variants of sJ-PAKE

This section describes the sRO-J-PAKE (Figure 7.5) and sCRS-J-PAKE (Figure 7.7) protocols in full detail. Moreover, we provide the security proof of both protocols in the same model as sJ-PAKE. We analyze both proofs in depth by going through games of sJ-PAKE and only point out the differences with sJ-PAKE. For simplicity, we leave the names and the number of games, bad events and reductions the same as the one in sJ-PAKE (Section 7.5). In addition, all changes in games and reductions of sRO-J-PAKE and sCRS-J-PAKE imply the changes in code-based proof. Most of the changes in the proof for sRO-J-PAKE and sCRS-J-PAKE occur in the reductions where we plug-in challenges differently compared to sJ-PAKE due to missing values X_1 and X_3. Lastly, we use the same assumptions as in sJ-PAKE.

7.6.1 sRO-J-PAKE

Description. Let H_0 be a full-domain hash mapping $\{0,1\}^*$ to $\mathbb{G}$. H_1 is a hash function from $\{0,1\}^*$ to $\{0,1\}^\kappa$. A function f is used to ensure that both parties sort values identically. Function D, basically, plays the role of $(X_1 X_3)$. In the first round, a client C generates X_1 and the proof of knowledge, π_1 for discrete log x_1 and transmits the identity, X_1 and π_1 to a Server S, while S does the same computation only with X_2 and π_2 and sends the identity, X_2 and π_2 to C. When both parties receive the message from the first round, they compute the common value $D = H_0(f(C, S, X_1, X_2))$. Then C generates $\alpha = (DX_2)^{x_1 \mathsf{pw}}$ and sends it to S, and at the same time, S generates $\beta = (DX_1)^{x_2 \mathsf{pw}}$ and sends it to C. After receiving α and β, both parties compute the key by the protocol $\mathsf{Key} = (\beta X_2^{-x_1 \mathsf{pw}})^{x_1}$ (resp. $\mathsf{Key} = (\alpha X_1^{-x_2 \mathsf{pw}})^{x_2}$). In the end, both parties hold the same session key $K = H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$.

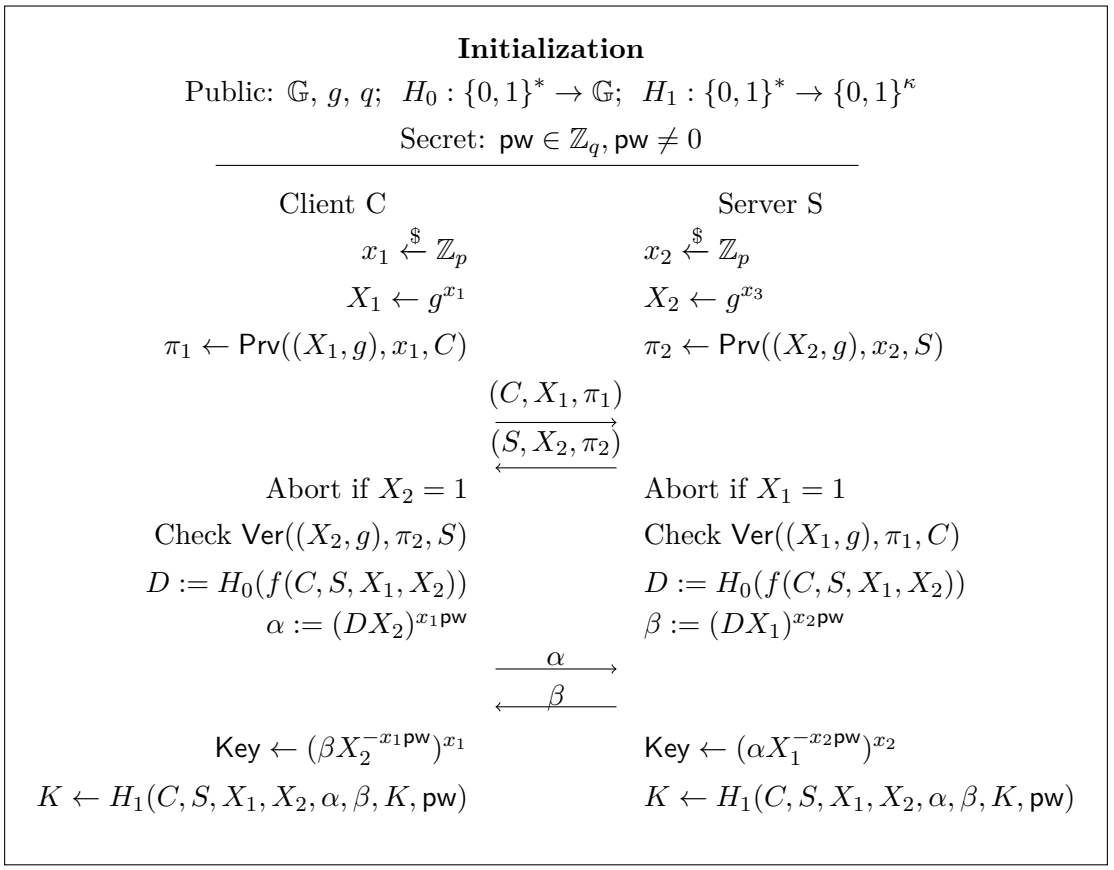

FIGURE 7.5: The sRO-J-PAKE protocol.

7.6.2 Security Game-based Proof of sRO-J-PAKE

Before the analysis of the game-based proof, we stress that adding full domain hash H_0 to the protocol means that our simulator will have to administer two random oracles: one oracle that responds to $H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$ queries[1] and another oracle that responds to $D = H_0(f(C, S, X_1, X_2))$ queries. Both oracles have to make sure that responses are consistent. We define RO to respond in case $\mathcal{A}$ asks $H_0(f(C, S, X_1, X_2))$ in Figure 7.6.

As already mentioned, function D in the sRO-J-PAKE protocol substitutes the X_1X_3 term. Thus, the only way the security proof will work is to simulate H_0 as $D = g^d$, where the simulator knows d. Since d is known and x_1 (resp. x_2) can be extracted from the ZK-PoK, the simulator will have all the information it needs to simulate the responses to any query that $\mathcal{A}$ asks. That being said, we give a formal proof of sRO-J-PAKE (Figure 7.5), and we only state the differences with respect to sJ-PAKE protocol. We

[1]This oracle is already defined and administrated in sJ-PAKE proof in Section 7.4.

For each fresh RO query $H_0(f(C, S, X_1, X_2))$ the simulator computes $D := g^d$, where $d \xleftarrow{\$} \mathbb{Z}_p$ and returns D to $\mathcal{A}$. The simulator administrates all the *query - response* by adding them to the list. If $\mathcal{A}$ already asked for H_0, the simulator simply retrieves the response D from the list and gives it to $\mathcal{A}$. The number of random oracle queries to H_0 we denote as n_{h0}.

FIGURE 7.6: Simulation of H_0

also refer the reader to [LST16], where the proof of RO-J-PAKE is displayed in detail, and we claim that sRO-J-PAKE has the same differences with sJ-PAKE as RO-J-PAKE has with J-PAKE. Furthermore, we stress that we modify all games by changing X_1, X_2, X_3 and X_4 to X_1, X_2, and π_1, π_2, π_3 and π_4 to π_1, π_2, and we add the second oracle that simulates random oracle queries for H_0. Now, we formally define security of sRO-J-PAKE in the following theorem:

Theorem 7.10. *Let sRO-J-PAKE be the protocol described in Figure 7.5. Take an algebraic RoR attacker $\mathcal{A}$ against sRO-J-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te}, n_{ro}, n_{ho} queries to* Send, Execute, Reveal, Corrupt, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that*

$$\begin{aligned}\mathrm{Adv}^{\mathtt{sRO-J-PAKE}}_{\mathcal{A}}() \leq\ & \frac{2n_{se}}{\mid D \mid} + \frac{(n_{se}+2n_{ex})^2}{q} \\ & + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK (defined in Chapter 2).*

Proof. **Game 0: Original protocol.**

$$\mathrm{Adv}^{\mathtt{sRO-J-PAKE}}_{\mathcal{A}}() = \mid \Pr[G_0 \Rightarrow \mathsf{T}] - \frac{1}{2} \mid$$

Game 1: Simulate ZK-PoK proofs.
This game is the same as in Game 1 of sJ-PAKE, except we simulate only proofs of knowledge of X_1 and X_2. Everything else stays the same. We have

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}()$$

Game 1.5: Extract discrete logs from adversarial proofs.
This game is the same as in Game 1.5 of sJ-PAKE, except we extract only from adversarial proofs of knowledge π_1 and π_2. Everything else stays the same. We have

$$|\Pr[G_1 \Rightarrow \mathsf{T}] - \Pr[G_{1.5} \Rightarrow \mathsf{T}]| \leq 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}()$$

Game 2: Force unique values
This game is the same as in Game 2 of sJ-PAKE except, we force the uniqueness of X_1 and X_2. We bound this change

$$\Pr[G_{1.5} \Rightarrow \mathsf{T}] - \Pr[G_2 \Rightarrow \mathsf{T}] \leq \frac{(n_{se} + 2n_{ex})^2}{q}$$

Game 3: Adding freshness. The description of this game is the same as in Game 3 of sJ-PAKE. There is no change in the bound.

$$\Pr[G_3 \Rightarrow \mathsf{T}] = \Pr[G_2 \Rightarrow \mathsf{T}]$$

Game 4: Randomize session keys for matching sessions.
The description of this game is the same as in Game 4 of sJ-PAKE, except for the natural change of X_1, X_2, X_3 and X_4 to X_1, X_2. This change implies a modification in the reduction to the CTGDH problem. We state the differences with the reduction described in Lemma 7.4:
Given the challenge $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$, we build an attacker $\mathcal{B}_4$ that finds $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$. We plug-in the challenge in $X_1 = X$ and $X_2 = Y$ while the values Z is embedded as the output of $H_0(C, S, X_1, X_2)$ and we set $\alpha = (D_{XY}D_{XZ})^{\mathsf{pw}_{cs}}$ and $\beta = (D_{XZ}D_{YZ})^{\mathsf{pw}_{cs}}$. Then, when the game finishes, the attacker guesses one query for which bad_4 might have occurred. If the guess was successful, it means that bad_4 occurred and $\mathcal{A}$ solved $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$, and $\mathcal{B}_4$ can recover g^{xyz} by computing:

$$g^{xyz} = \mathsf{Key}^{\frac{1}{\mathsf{pw}_{cs}}}$$

Therefore, we have the same bound:

$$|\Pr[G_3 \Rightarrow \mathsf{T}] - \Pr[G_4 \Rightarrow \mathsf{T}]| \leq n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Game 4.5: Randomize alpha/beta for Execute queries.
The description of this game is the same as in Game 4.5 of sJ-PAKE. The only change we do is in the reduction to the DDH problem when randomizing both α and β in Games 4.5.1. and 4.5.2. We state the differences with the reductions described in Lemma 7.5 and Lemma 7.6:
Similar to Lemma 7.5 (when randomizing α), $\mathcal{B}_{4.5.1}$ is given a generalized challenge $(X_1, ..., X_{n_{se}}, Y_1, ..., Y_{n_{se}}, Z_1, ..., Z_{n_{se}})$, and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge and for every $\mathsf{Execute}$ query sets $X_1 = X_k$, $X_2 = g^{x_2}$ and $\alpha = (X_k^{x_2} Z_k)^{\mathsf{pw}_{cs}}$ where $D = Y_k$.
Similar to Lemma 7.5 (when randomizing β), $\mathcal{B}_{4.5.2}$ is given a generalized challenge $(X_1, ..., X_{n_{se}}, Y_1, ..., Y_{n_{se}}, Z_1, ..., Z_{n_{se}})$, and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge and for every $\mathsf{Execute}$ query sets $X_1 = g^{x_1}$, $X_2 = X_k$ and $\beta = (Y_k^{x_1} Z_k)^{\mathsf{pw}_{cs}}$ where $D = Y_k$. The bounds for Game 4.5.1 and Game 4.5.2 do not change with respect to the same games as in s-JPAKE:

$$\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

$$\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Game 5: Randomize session keys for Send queries. The description of this game is the same as in Game 5 of sJ-PAKE, so the bound does not change:

$$|\Pr[G_{4.5.2} \Rightarrow \mathsf{T}] - \Pr[G_5 \Rightarrow \mathsf{T}]| \leq \Pr[G_5 \Rightarrow \mathsf{bad}_5]$$

Game 6: Detect duplicates.
The description of this game is the same as in Game 6 of sJ-PAKE, except for modification in the reduction to the CSqDH problem. We state the differences with the reduction described in Lemma 7.7:
Given the challenge X, we build an attacker $\mathcal{B}_6$ that finds solution to the CSqDH problem. Let us say that $\mathcal{B}_6$ plugs-in the challenge in the ℓ^{th} session on the client's side, in $X_1 = X$ and it sets $\alpha = X^{(x_2' + d)\mathsf{pw}_{cs}}$, where x_2' is the witness extracted from adversarial proof π_2' and d is the discrete log of D which represents the output of $H_0(C, S, X_1, X_2)$. Then, when the game finishes, $\mathcal{B}_6$ guesses one query for which bad_6 might have occurred. If the guess was successful it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_6$ can recover g^{x^2} in the following way:

$$\mathsf{Key}_1 = (\frac{\beta'}{X^{x_2'\mathsf{pw}_{cs}}})^x = (\frac{(XD)^{\mathsf{pw}_1}}{X^{x_2'\mathsf{pw}_{cs}}})^x = X^{dx_2'\mathsf{pw}_1} g^{x^2 x_2'(\mathsf{pw}_1 - \mathsf{pw}_{cs})} \quad (1)$$

$$\mathsf{Key}_2 = (\frac{\beta'}{X^{x_2'\mathsf{pw}_{cs}}})^x = (\frac{(XD)^{\mathsf{pw}_2}}{X^{x_2'\mathsf{pw}_{cs}}})^x = X^{dx_2'\mathsf{pw}_2} g^{x^2 x_2'(\mathsf{pw}_2 - \mathsf{pw}_{cs})} \quad (2)$$

where x is unknown. From (1) and (2) follows:

$$g^{x^2} = (\frac{\mathsf{Key}_1}{\mathsf{Key}_2 X^{dx_2'(\mathsf{pw}_1 - \mathsf{pw}_2)}})^{\frac{1}{x_2'(\mathsf{pw}_1 - \mathsf{pw}_2)}}$$

We do the same procedure on the server's side and plug-in the challenge in $X_2 = X$ and set $\beta = X^{(x_2'+d)\mathsf{pw}_{cs}}$. The bound and other remarks do not change:

$$\Pr[G_5 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_6 \Rightarrow \mathsf{bad}_5] + n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

Game 7: Add algebraic representation.
The description of this game is the same as in Game 7 of sJ-PAKE. Thus, the bound does not change:

$$\Pr[G_6 \Rightarrow \mathsf{bad}_5] = \Pr[G_7 \Rightarrow \mathsf{bad}_5]$$

Game 8: Randomizing α and β for Send queries.
The description of this game is the same as in Game 8 of sJ-PAKE. The only change we do is in the reduction to the DSqDH problem when bounding bad cases of α and β in Game 7-8.1. We state the differences with the reduction described in Lemma 7.8:
Given the challenge (X, Y), we build an attacker $\mathcal{B}_{7-8.m.1}$ that submits $b = 1$ if (X, Y) is a real DSqDH tuple and $b = 0$ if it is a random DSqDH tuple. Further, $\mathcal{B}_{7-8.m.1}$ uses a hybrid argument, and for a fresh m^{th} session, it plugs-in the challenge for the client-side $X_1 = X$ and sets $\alpha = X^{(x_2'+d)\mathsf{pw}_{cs}}$ where d is selected by the attacker and x_2' is extracted from adversarial proof of knowledge. We consider having algebraic adversary $\mathcal{A}$, and whenever it sends β', it also sends (a, b, c), so in case of Corrupt, we rewrite $\beta = g^a X_1^b \alpha^c$. Then we check:

(a) If $X_1^b \alpha^c \neq X_1^{x_2'\mathsf{pw}_{cs}}$ the key is computed as $\mathsf{Key} = X_1^a Y^{(b+d\mathsf{pw}_{cs})} = \alpha^{\frac{a}{(x_2'+d)\mathsf{pw}_{cs}}} Y^{(b+d\mathsf{pw}_{cs})}$. This means if $Y = g^{x^2}$ then the key is real and random otherwise.

(b) If $X_1^b \alpha^c = X_1^{x_2'\mathsf{pw}_{cs}}$, we embed α in the key and compute it as $\mathsf{Key} = \alpha^{\frac{a}{(x_2'+d)\mathsf{pw}_{cs}}}$.
We say $\mathcal{A}$ interpolates between the two games, meaning, if $Y = g^{x^2}$ then it is playing Game 7, and otherwise, it is playing Game 7-8.m.1. The analysis is the same on the

server's side. Furthermore, the bounds and other remarks for Game 7-8.1 and Game 7-8.2 do not change with respect to the same games in s-JPAKE. Thus, we have:

$$\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}] \leq 2\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

and

$$|\Pr[G_{7-8.m.1} \Rightarrow T] - \Pr[G_{7-8.m.2} \Rightarrow T]| \leq \Pr[G_8 \Rightarrow \mathsf{bad}_5]$$

Game 9: Perfect forward secrecy.
The description of this game is the same as in Game 9 of sJ-PAKE, except for modification in the reduction to the CSqDH problem. We state the differences with the reduction described in Lemma 7.9:
Given the challenge X, we build an attacker $\mathcal{B}_9$ that finds solution to CSqDH problem. Let us say that $\mathcal{B}_9$ plugs in the challenge in the ℓ^{th} session on the client's side, in $X_1 = X$ and it sets $\alpha = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. When $\mathcal{A}$ sends β' with (a, b, c), we rewrite it as $\beta' = g^a X_1^b \alpha^c$. Then, when the game finishes, $\mathcal{B}_9$ guesses one query for which bad_9^1 might have occurred. If the guess was successful, it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_9$ can recover g^{x^2} in the following way:

$$g^{x^2} = \left(\frac{\mathsf{Key}}{X^{(a+wc)}}\right)^{\frac{1}{b - x_2' \mathsf{pw}_{cs}}}$$

We do the same procedure on the server-side and plug in the challenge for $X_2 = X$ and set $\beta = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. The bound and other remarks do not change, so we have

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^1] \leq n_{se} n_{ro} \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()$$

□

7.6.3 sCRS-J-PAKE

Description. At the beginning of the protocol, sCRS-J-PAKE sets up a common reference string U (as in CRS-J-PAKE). Basically, U plays the role of $(X_1 X_3)$. The first round is the same as in sRO-J-PAKE (RO-J-PAKE), with the X_1, X_3 and proofs π_1 and π_3 eliminated. In the second round, after receiving the first message, a Client computes $\alpha = (UX_2)^{x_1 pw}$, and a Server $\beta = (UX_1)^{x_2 pw}$ and they exchange α and β. Both parties compute the key as $\mathsf{Key} = (\beta X_2^{-x_1 \mathsf{pw}})^{x_1}$ (resp. $\mathsf{Key} = (\alpha X_1^{-x_2 \mathsf{pw}})^{x_2}$) and hold the same session key, $K = H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$.

Initialization

Public: $\mathbb{G}$, g, q, $U \in \mathbb{G}$; $H_1 : \{0,1\}^* \to \{0,1\}^\kappa$

Secret: $\mathsf{pw} \in \mathbb{Z}_q, \mathsf{pw} \neq 0$

Client C		Server S
$x_1 \xleftarrow{\$} \mathbb{Z}_p$		$x_2 \xleftarrow{\$} \mathbb{Z}_p$
$X_1 \leftarrow g^{x_1}$		$X_2 \leftarrow g^{x_3}$
$\pi_1 \leftarrow \mathsf{Prv}((X_1, g), x_1, C, U)$		$\pi_2 \leftarrow \mathsf{Prv}((X_2, g), x_2, S, U)$
	$\xrightarrow{(C, X_1, \pi_1)}$	
	$\xleftarrow{(S, X_2, \pi_2)}$	
Abort if $X_2 = 1$		Abort if $X_1 = 1$
Check $\mathsf{Ver}((X_2, g), \pi_2, S, U)$		Check $\mathsf{Ver}((X_1, g), \pi_1, C, U)$
$\alpha := (UX_2)^{x_1 \mathsf{pw}}$		$\beta := (UX_1)^{x_2 \mathsf{pw}}$
	$\xrightarrow{\alpha}$	
	$\xleftarrow{\beta}$	
$\mathsf{Key} \leftarrow (\beta X_2^{-x_1 \mathsf{pw}})^{x_1}$		$\mathsf{Key} \leftarrow (\alpha X_1^{-x_2 \mathsf{pw}})^{x_2}$
$K \leftarrow H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		$K \leftarrow H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$

FIGURE 7.7: The sCRS-J-PAKE protocol.

7.6.4 Security Game-based Proof of sCRS-J-PAKE

We formally define the security of the sCRS-J-PAKE in the following theorem:

Theorem 7.11. *Let sCRS-J-PAKE be the protocol described in Figure 7.7. Take an algebraic RoR attacker $\mathcal{A}$ against sCRS-J-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te} queries to* Send, Execute, Reveal, Corrupt, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that*

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}^{\mathtt{sCRS-J-PAKE}}() \leq \quad & \frac{2n_{se}}{\mid D \mid} + \frac{(2n_{se}+4n_{ex})^2}{q} \\ & + \mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{uzk}}() + 2n_{se}\mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{ext}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}_{\mathcal{B}_4}^{\mathtt{CTGDH}}() + \mathrm{Adv}_{\mathcal{B}_{4.5.1}}^{\mathtt{DDH}}() + \mathrm{Adv}_{\mathcal{B}_{4.5.2}}^{\mathtt{DDH}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}_{\mathcal{B}_6}^{\mathtt{CSqDH}}() + 2n_{se}\mathrm{Adv}_{\mathcal{B}_{7-8.1}}^{\mathtt{DSqDH}}() + n_{se}n_{ro}\mathrm{Adv}_{\mathcal{B}_9}^{\mathtt{CSqDH}}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK (defined in Chapter 2).*

Proof. The proof of sCRS-J-PAKE is the same as sRO-J-PAKE in Section 7.6.2 with the differences:

- We do not need an additional random oracle to simulate H_0. Instead, we have CRS U, known to both parties.
- In the reduction described in Lemma 7.4, we embed a challenge Z in place of U in α (resp. β)
- We add Game 3.5 (explained below), where we explicitly save a discrete log of U during the execution, needed for simulation in later hops.

Game 3.5: Keep the discrete log of the public parameter.
During the initialization phase described in Figure 7.8, the discrete log u is saved as a record (U, u) on the list for future hops. More precisely, the simulator can retrieve the record (U, u) from the list at any time of the execution of the protocol.

Initialization:
Choose $u \xleftarrow{\$} \mathbb{Z}_p$.
Compute $U = g^u$ and save the record (u, U) to the list.
For $C \in \mathcal{C}$, $S \in \mathcal{S}$: $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\kappa)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return U, CRS, C, S;

FIGURE 7.8: Initialization phase for sCRS-J-PAKE

□

7.7 Efficiency Analysis of J-PAKE, sJ-PAKE and All its Variants

In this section, we only focus on J-PAKE and its variants and provide a concrete analysis of complexity that also is a result of this work. Conversely, a comparison of J-PAKE with other known PAKE protocols is shown in [ABM15]. We mainly rely on empirical results and performance analysis of J-PAKE, RO-J-PAKE, CRS-J-PAKE shown in [LST16] and convey the same analysis on sJ-PAKE, sRO-J-PAKE and sCRS-J-PAKE. Furthermore, we explicitly highlight the differences in computation costs, given in Table 7.1, and conclude which protocol(s) are the most efficient one(s). In terms of communication, J-PAKE weighs 6 groups elements plus 12 ZK-PoK scalars in $\mathbb{Z}_q$ and 28 exponentiations in terms of computation which makes J-PAKE the least efficient among them all. Then, sJ-PAKE follows with 2 ZK-PoKs dropped in the second round, resulting in 22 exponentiations. Then, RO-J-PAKE and CRS-J-PAKE drop 2 ZK-PoKs and 2 group elements in the first round, counting a total of 20 exponentiation. Here, we stress that even though the difference in computation with sJ-PAKE is only two exponentiation, sJ-PAKE is a more desirable "lightweight" version, as it requires minimal changes to the current J-PAKE implementation. In contrast, RO-J-PAKE and CRS-J-PAKE implementations require more changes due to the additional trusted setup for the CRS-J-PAKE and additional cost of a hash H_0 for the RO-J-PAKE. Our final analysis includes sRO-J-PAKE and sCRS-J-PAKE. The total computation cost drops to only 14 exponentiations when dropping two ZK-PoKs in the first round (along with two group elements) and dropping two ZK-PoKs in the second round. Compared to J-PAKE, sRO-J-PAKE and sCRS-J-PAKE have a significant drop in exponentiation, making the proof of sJ-PAKE beneficial as it suggests that sRO-J-PAKE and sCRS-J-PAKE can be

TABLE 7.1: Comparison of Complexity of all J-PAKE(s) and the corresponding assumptions used to prove their security against active attackers.

Protocol	**Complexity**		**Hardness Ass.**[1]	**Forward secrecy**
	Communication[2]	**Computation**		
J-PAKE	$6 \times \mathbb{G} + 12 \times \mathbb{Z}_q$	28 | p |-bit exp [3]	DSqDH	PFS[4]
RO-J-PAKE	$4 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	20 | p |-bit exp+$2H_0$[5]	DSqDH	PFS
CRS-J-PAKE	$4 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	20 | p |-bit exp	DSqDH	PFS
sJ-PAKE	$6 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	22 | p |-bit exp	CSqDH	PFS
sRO-J-PAKE	$4 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	14 | p |-bit exp+$2H_0$	CSqDH	PFS
sCRS-J-PAKE	$4 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	14 | p |-bit exp	CSqDH	PFS

[1] DSqDH stands for Decisional Square Diffie-Hellman, while CSqDH stands for Computational Square Diffie-Hellman
[2] $\mathbb{G}$ denotes a group element, $\mathbb{Z}_q$ a scalar
[3] exp. denotes exponentiation in $\mathbb{G}$. ZK-PoK costs three exponentiation each (one to create and two to verify)
[4] Perfect Forward Secrecy
[5] $H_0 : \{0,1\}^* \rightarrow \mathbb{G}$

proven secure in the same manner. To confirm, we lay out the security proof of sRO-J-PAKE and sCRS-J-PAKE in Section 7.6. Furthermore, with the sJ-PAKE protocol and its security proof, there is a stronger motivation for implementing sRO-J-PAKE and sCRS-J-PAKE in practice.

No great discovery was ever made without a bold guess.

Sir Isaac Newton

8

Future Work and Conclusion

8.1 Future Work

For each work, presented in this dissertation, we provide future directions:

1. Since zkPAKE protocol core design is flawed beyond repair and many mature PAKE alternatives already exist, we do not pursue further study to improve the zkPAKE protocol.

2. We leave the possibility for a full security proof of HoneyPAKEs and perhaps, generalizing the transformation of any PAKE to HoneyPAKEs composition.

3. We plan to extend the research of our latest result sJ-PAKE protocol (presented in Chapter 7) and follow two directions: in one, we try to prove sJ-PAKE secure in the UC relaxed model [ABB+20], and in the other, we propose sJ-PAKE with confirmation codes that we can prove secure in the stronger (UC) model [CHK+05].

Finally, in the following sections, we continue discussing the directions concerning sJ-PAKE protocol with confirmation codes.

8.1.1 sJ-PAKE in UC

It is unclear whether we can prove sJ-PAKE (sRO-J-PAKE and CRS-J-PAKE, respectively) in a UC framework setting. We stress that the proof of sJ-PAKE resembles the proof for perfect forward secrecy of SPAKE2 [AB19], which has security proof in the UC

relaxed model. However, unlike SPAKE2, proving sJ-PAKE in the relaxed UC seems like a more difficult task due to heavier use of algebraic adversaries (see Game 8 and Game 9 in Section 7.4). Furthermore, without assuming algebraic adversary in Game 6, it seems that sJ-PAKE cannot be proven secure in the relaxed UC at all.[1] For now, we pay more attention to adding confirmation codes to the sJ-PAKE protocol. We think there is already substantial research out there on security proofs for key-exchange protocols with confirmation codes than on security proofs with algebraic adversaries in the UC and UC relaxed model.

8.1.2 sJ-PAKE with Confirmation Codes

We propose a new protocol, sJ-PAKE with confirmation codes, that might have a tighter and more straightforward security proof than sJ-PAKE, and a better chance to be proven secure in the UC model. Therefore, we anchor our analysis of sJ-PAKE with confirmation codes mostly on sJ-PAKE and its proof. Furthermore, our idea is to have two variations of sJ-PAKE with confirmation codes. In the first one, Figure 8.2, we add the first code k_1 on the server's side, just after computing β. Then the client would verify k_1 and compute the Key and the confirmation code k_2. Note that this would break the symmetry of the protocol and might additionally complicate the proof. In the second version (Figure 8.1), we add the third round, where both parties exchange confirmation codes, without disrupting the symmetry of the protocol. We stress that each party computes the session key in both versions after confirmation codes are verified. In addition, both versions are interesting from three aspects:

1. With confirmation codes, we gain explicit authentication for both parties. This is very useful for two reasons: First, when having confirmation codes, usually one obtains *tighter* reductions; thus, we can gain better security bounds. Secondly, protocols that obtain session keys from PAKEs would receive information from explicitly authenticated PAKE whether the session is successfully completed. This is a beneficial property for the following reason: consider a PAKE protocol that provides only implicit authentication. This would mean that an adversary who is making a wrong password guess (thus holding a non-matching session key) would still get a chance to interact with the protocol communicating with PAKEs, which would not be the case if explicitly authenticated PAKE would be used.

2. Assuming that the new protocol initially satisfies weak forward secrecy, when combined with confirmation codes, the protocol then satisfies perfect forward secrecy (PFS) [Mac02, BPR00, Sho99].

[1] If the adversary is algebraic in Game 6, we can extract the password guesses from the second-round messages.

3. By introducing confirmation codes, we hope to omit algebraic adversaries presented in Game 8 and Game 9 of the sJ-PAKE protocol. Without algebraic adversaries and having access to another password commitment before the session key is generated, we believe we could prove perfect forward secrecy more straightforwardly.

4. Minimal change to the implementation. For now, there is no implementation of sJ-PAKE. However, once the implementation becomes public, only a few lines of code should be added to obtain a new variant of sJ-PAKE with confirmation codes.

We stress that disrupting the symmetry as in version 2 (Figure 8.2) would mean that there is an order in the message flow to send the message in each round. However, it might lead to a more complicated proof.

Sketch of the proof. Usually, confirmation codes are considered to improve the protocol in question, which is precisely our motivation for sJ-PAKE with confirmation codes. However, these improvements usually increase the number of rounds, which means that the new protocols have higher communication and latency costs than sJ-PAKE. [Mac02, Kra05]. Nevertheless, we believe that most of the proof of sJ-PAKE with confirmation codes would require minor tweaks with respect to the existing proof of sJ-PAKE. For instance, in Game 8 and Game 9, we would not have problematic cases due to additional information that confirmation codes provide and verify before computing the keys. We think that we could achieve perfect forward secrecy in Game 5, and probably Game 9 would be non-existent.
To conclude our analysis, both versions of sJ-PAKE with confirmation codes would transitively yield new variants (as sRO-J-PAKE and sCRS-J-PAKE in sJ-PAKE). Furthermore, we believe that security proof of both versions of sJ-PAKE with confirmation codes would be simpler than that of sJ-PAKE. Lastly, we underline the possibility of proving sJ-PAKE with confirmation codes in the UC relaxed model [ABB$^+$20].

Initialization		
Public information:	$\mathbb{G}, g, q, \sigma \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$;	$H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^\kappa$
Secret information:	$\mathsf{pw} \in \mathbb{Z}_p, \mathsf{pw} \neq 0$	

C		**S**
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$		$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$
$X_1 \leftarrow g^{x_1}, X_2 \leftarrow g^{x_2}$		$X_3 \leftarrow g^{x_3}, X_4 \leftarrow g^{x_4}$
Generate π_1, π_2		*Generate* π_3, π_4
	$(C, X_1, X_2, \pi_1, \pi_2) \longrightarrow$	
	$\longleftarrow (S, X_3, X_4, \pi_3, \pi_4)$	
Verify π_3, π_4, X_4		*Verify* π_1, π_2, X_2
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}}$		$\beta \leftarrow (X_1 X_2 X_3)^{x_4 \mathsf{pw}}$
	$\alpha \longrightarrow$	
	$\longleftarrow \beta$	
$\mathsf{Key} \leftarrow (\beta X_4^{-x_2 \mathsf{pw}})^{x_2}$		$\mathsf{Key} \leftarrow (\alpha X_2^{-x_4 \mathsf{pw}})^{x_4}$
$k_1 \leftarrow H_1(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		$k_2 \leftarrow H_2(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$
	$k_1 \longrightarrow$	
	$\longleftarrow k_2$	
Verify k_2		*Verify* k_1
$sk \leftarrow H_3(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		$sk \leftarrow H_3(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$

FIGURE 8.1: Version 1: sJ-PAKE with confirmation codes

Initialization

Public information:	$\mathbb{G}, g, q, \sigma \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$;	$H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^\kappa$
Secret information:	$\mathsf{pw} \in \mathbb{Z}_p, \mathsf{pw} \neq 0$	

C		**S**
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$		$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$
$X_1 \leftarrow g^{x_1}, X_2 \leftarrow g^{x_2}$		$X_3 \leftarrow g^{x_3}, X_4 \leftarrow g^{x_4}$
Generate π_1, π_2		*Generate* π_3, π_4
	$(C, X_1, X_2, \pi_1, \pi_2) \longrightarrow$	
	$\longleftarrow (S, X_3, X_4, \pi_3, \pi_4)$	
Verify π_3, π_4, X_4		*Verify* π_1, π_2, X_2
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}}$		$\beta \leftarrow (X_1 X_2 X_3)^{x_4 \mathsf{pw}}$
	$\alpha \longrightarrow$	
		$\mathsf{Key} \leftarrow (\alpha X_2^{-x_4 \mathsf{pw}})^{x_4}$
		$k_1 \leftarrow H_1(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$
	$\longleftarrow \beta, k_1$	
Verify k_1		
$\mathsf{Key} \leftarrow (\beta X_4^{-x_2 \mathsf{pw}})^{x_2}$		
$k_2 \leftarrow H_2(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		
	$k_2 \longrightarrow$	
		Verify k_2
$sk \leftarrow H_3(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$		$sk \leftarrow H_3(C, S, X_1, .., X_4, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$

FIGURE 8.2: Version 2: sJ-PAKE with confirmation codes

8.2 Conclusion and Final Remarks

So far, PAKE protocols have been studied intensively as a cryptographic primitive and have successfully been used in practice, e.g., for Internet authentication and e-passports. This dissertation not only provides new protocols with clear advantages and provable security, but also presents novel scenarios where one could benefit from PAKEs. Furthermore, our work also confirms that the protocol that comes with the proof provides substantial confidence that it is secure to use for real-world deployment and set a cornerstone for standardization. Finally, we lay out three different works that offer different lines of research on PAKEs and we summarize them in the following points:

1. We showed an efficient offline dictionary attack in practice on the PAKE protocol, zkPAKE. We stress that cryptographers should be more careful when designing a protocol and ensure that it meets its formal security requirements. Furthermore, we underline that security proof is necessary when proposing a new protocol.

2. We have presented a new login scenario – a way of merging PAKE-based access control with Honeywords to get the benefits of both:

   - Intrinsic protection of the password during the login phase.
   - Detection of attempts to exploit the compromise of a password file.

   We also demonstrated a variant that incorporates a two-factor mechanism in a very natural way, where the token-generated nonce plays the role of the secondary password. Further, we presented a variant of the protocol in which the honey server S does not directly learn the index of the correct (hashed) password.

3. We have proposed a lightweight version of J-PAKE, sJ-PAKE, that avoids the NIZK proofs of knowledge in the second round, thus significantly improving the efficiency both in terms of computation and communication. Moreover, we explicitly show the challenging parts of the proof (Game 8 and Game 9) where we use algebraic adversaries to prove perfect forward secrecy. At the first sight, sJ-PAKE seems to fill the requirements to be proven secure in the UC relaxed framework [ABB$^+$20]. Still, it seems that without heavier use of algebraic adversaries achieving the sJ-PAKE secure in any UC framework is not possible. In addition, we provide arguments that the efficiency gains of earlier lightweight variants of J-PAKE, RO-J-PAKE and CSR-J-PAKE, that reduce the number of NIZK proofs in the first round can be combined with those of sJ-PAKE, thus providing ultra-light variants, and we provide game-based proofs of security for these variants as well.

A

Supplemental material

A.1 Game-based code

A.1.1 Games and adversaries for the proof of Theorem 7.1.

Game 0: Original Protocol

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\kappa)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_1, g), x_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_2, g), x_2, C)$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_3, g), x_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_4, g), x_4, S)$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3 . X_4')^{x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X_1' X_2' X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_1, g), x_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_2, g), x_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_3, g), x_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_4, g), x_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.1: Game 0

Game 1: Simulate ZK-PoK proofs.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X_1' X_2' X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1+x_3+x_4)x_2 pw_{cs}}$;
$\beta = g^{(x_1+x_2+x_3)x_4 pw_{cs}}$;
$\mathsf{Key} = g^{(x_1+x_3)x_2 x_4 pw_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

$\mathbf{H}(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.2: Game 1

Reduction for Game 1

Initialize (crs)
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\boxed{\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C);}$
$\boxed{\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\boxed{\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S);}$
$\boxed{\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X_1' X_2' X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\boxed{\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C);}$ $\boxed{\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C);}$
$\boxed{\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S);}$ $\boxed{\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1+x_3+x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1+x_2+x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1+x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

$\mathbf{H}(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = 1$ return 1;
Else return 0;
Abort.

FIGURE A.3: Reduction for Game 1

Game 1.5: Extract discrete logs from adversarial proofs.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List}_{\mathsf{Ext}} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
$bad_{1.5} = \mathsf{T}$; For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $bad_{1.5} = \mathsf{T}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $bad_{1.5} = \mathsf{T}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2(S,i,C,X'_1,X'_2, π'_1, π'_2)
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $bad_{1.5} = \mathsf{T}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $bad_{1.5} = \mathsf{T}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$Key \leftarrow (\frac{\beta'}{X'^{x_2 \mathsf{pw}_{cs}}_4})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X'^{x_4 \mathsf{pw}_{cs}}_2})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi^i_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi^i_U) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b' = b$;

FIGURE A.4: Game 1.5

Reduction for Game 1.5

Initialize $(\mathsf{crs}, \mathsf{td}_e)$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C)$;
$\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S)$;
$\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_3', \pi_3'\}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ do $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_4', \pi_4'\}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ Abort;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_1', \pi_1'\}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_2', \pi_2'\}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, T)$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, T)$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(X', \pi') \xleftarrow{\$} \mathsf{List}_{\mathsf{Ext}}$ do $\mathsf{NIZK.Finalize}(X', \pi')$;
Return $b = b'$;

FIGURE A.5: Reduction for Game 1.5

Game 2: Force unique values

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
bad_2=F;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
If $X_1 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.6: Game 2

Game 3: Adding freshness.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\boxed{\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr};}$
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\boxed{\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr};}$
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\boxed{\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T});}$
$\boxed{\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr};}$
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\boxed{\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T});}$
$\boxed{\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr};}$
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi^i_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\boxed{\forall (j, V) \text{ s.t. } (\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid}) \text{ do } \pi^j_V.\mathsf{fr} = \mathsf{F};}$
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.7: Game 3

Game 4: Randomize session Keys for passive adversaries.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$T_e \leftarrow \{\}$; $bad_4 = \mathsf{F}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ Abort;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \leftarrow g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta \leftarrow g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$\mathsf{Key}^* \leftarrow g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
If $(\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}) \in T$ then $bad_4 = \mathsf{T}$;
$T_e \leftarrow T_e \cup \{\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T_e$ then $bad_4 = \mathsf{T}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.8: Game 4

Reduction for Game 4

Initialize $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$T_e \leftarrow \{\}$; $T_4 \leftarrow \{\}$; $bad_4 = \mathsf{F}$
$l \xleftarrow{\$} \{1, \ldots n_{ex}\}$; $p \leftarrow 0$; $r \leftarrow 0$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$r \leftarrow r + 1$;
If $r \neq l$
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $r = l$
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$r \leftarrow r + 1$;
If $r \neq l$
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $r = l$
$X_3 \leftarrow Y$; $x_3 \leftarrow \perp$; $X_4 \leftarrow Z$; $x_4 \leftarrow \perp$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
Else
  $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
Else
  $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$p \leftarrow p + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $p = l$
  $X_2 \leftarrow X$; $X_3 \leftarrow Y$; $X_4 \leftarrow Z$;
  $x_2 \leftarrow \perp$; $x_3 \leftarrow \perp$; $x_4 \leftarrow \perp$;
Else
  $x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
  $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
If $x_2 = \perp \vee x_3 = \perp \vee x_4 = \perp$
  $\alpha \leftarrow (D_{XY} D_{XZ} X^{x_1})^{\mathsf{pw}_{cs}}$
  $\beta \leftarrow (D_{XZ} D_{YZ} Z^{x_1})^{\mathsf{pw}_{cs}}$
  $\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
  If $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then $T_4 \leftarrow T_4 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}\}$;
Else
  $\alpha \leftarrow g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
  $\beta \leftarrow g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
  $\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
  $\mathsf{Key}^* \leftarrow g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
  If $(\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}) \in T$ then $bad_4 = \mathsf{T}$;
  Else $T_e \leftarrow T_e \cup \{\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T_e$ then $T_4 \leftarrow T_4 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, pw] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_4$ do $\mathcal{B}_4.\mathsf{Finalize}(g^{xyz})$;
Abort.

FIGURE A.9: Reduction for Game 4

Game 4.5.1 Randomize α

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\boxed{\alpha \xleftarrow{\$} \mathcal{G};}$
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.10: Game 4.5.1

Reduction for Game 4.5.1.

Initialize $(X_1, \ldots X_{ex}, Y_1, \ldots, Y_{ex}, Z_1, \ldots, Z_{ex})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$k \leftarrow 0$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4') x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$k = k + 1$;
$X_1 \leftarrow X_k$; $x_1 \leftarrow \perp$;
$X_2 \leftarrow Y_k$; $x_2 \leftarrow \perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \leftarrow (Z_k X_2^{x_3 + x_4})^{\mathsf{pw}_{cs}}$;
$\beta \leftarrow X_2^{x_4 \mathsf{pw}_{cs}} g^{(x_1 + x_2) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = 1$ return 1;
Else return 0;
Abort.

FIGURE A.11: Reduction for Game 4.5.1

Game 4.5.2 Randomize β

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2($C, i, S, X_3', X_4', \pi_3', \pi_4'$)
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\boxed{\beta \xleftarrow{\$} \mathcal{G};}$
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.12: Game 4.5.2

Reduction for Game 4.5.2

Initialize $(X_1, ...X_{ex}, Y_1, ..., Y_{ex}, Z_1, ..., Z_{ex})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$k \leftarrow 0$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$k = k + 1$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$X_3 \leftarrow X_k$; $x_3 \leftarrow \perp$;
$X_4 \leftarrow Y_k$; $x_4 \leftarrow \perp$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$;
$\beta \leftarrow (Z_k X_4^{x_1 + x_2})^{\mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$T_e \leftarrow T_e \cup \{\mathsf{sid}, k, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = b$ return 1;
Else return 0;
Abort.

FIGURE A.13: Reduction for Game 4.5.2

Game 5: Randomize session Keys for Send queries.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$T_s \leftarrow \{\}$; bad_5=F;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else
  $\forall(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then bad_5=T; Abort.
  $K \xleftarrow{\$} \mathcal{K}$;
  $T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_2'}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else
  $\forall(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then bad_5=T; Abort.
  $K \xleftarrow{\$} \mathcal{K}$;
  $T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  If $T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then bad_5=T; Abort.
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall(j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

FIGURE A.14: Game 5

Game 6: Detect duplicates

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
bad_5=F; $bad_6 = \mathsf{F}$; $T_6 \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else If $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else If $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}}})^{x_2}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $\exists\ \mathsf{pw} \neq \mathsf{pw}'$
  $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \xleftarrow{\$} T_6 \wedge (\mathsf{sid}, \mathsf{Key}', \mathsf{pw}') \xleftarrow{\$} T_6$ do $bad_6 = \mathsf{T}$;
If bad_6=F $\wedge\ T_6 \neq \emptyset$ then bad_5=T;
Return $b = b'$;

FIGURE A.15: Game 6

Reduction for Game 6.

Initialize (X)
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
$T_6 \leftarrow \{\}$; $T_{\mathsf{bad}} \leftarrow \{\}$;
$l \xleftarrow{\$} \{1, \ldots n_{se}\}$; $r \leftarrow 0$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $r = l$
$X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
Else
$x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 = g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $r = l$
$X_4 \leftarrow X$; $x_4 \leftarrow \perp$;
Else
$x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
$x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
$x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $x_2 = \perp$
$\alpha \leftarrow X_2^{(x_1 + x_3' + x_4')\mathsf{pw}_{cs}}$;
Else
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
$x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
$x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $x_4 = \perp$
$\beta \leftarrow X_4^{(x_1' + x_2' + x_3)\mathsf{pw}_{cs}}$;
Else
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
If $x_2 \neq \perp$
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
If $x_2 = \perp$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then Abort. ("Wrong guess")
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
If $x_2 \neq \perp$
$\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $x_2 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
If $x_4 \neq \perp$
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
If $x_4 = \perp$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then Abort. ("Wrong guess")
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
If $x_4 \neq \perp$
$\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}}})^{x_4}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$
If $x_4 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
If $x_2 = \perp \vee x_4 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $\exists\ \mathsf{pw} \neq \mathsf{pw}'$
$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \xleftarrow{\$} T_{\mathsf{bad}} \wedge (\mathsf{sid}, \mathsf{Key}', \mathsf{pw}') \xleftarrow{\$} T_{\mathsf{bad}}$ do
$\mathcal{B}_6.\mathsf{Finalize}(g^{x^2})$; Abort.
If $T_{\mathsf{bad}} \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then bad_5=T;
Return $b = b'$;

FIGURE A.16: Reduction for Game 6

Game 7: Adding algebraic representation.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $T_6 \leftarrow \{\}$;
bad_5=F;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3$(C, i, S, \beta', \boxed{\mathsf{alg}'})$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else $\forall \mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K, \boxed{\mathsf{alg}}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K, \boxed{\mathsf{alg}'})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3$(S, i, C, \alpha', \boxed{\mathsf{alg}'})$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \boxed{\mathsf{alg}}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \boxed{\mathsf{alg}'})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $\forall T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, Key, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, Key, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, Key, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then bad_5=T;
Return $b = b'$;

FIGURE A.17: Game 7

Game 7-8.1: Randomizing α and β.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$; $t \leftarrow 0$;

For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$t \leftarrow t+1$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S,i,C)
If $\pi_S^i \neq \perp$ return $\perp$;
$t \leftarrow t+1$;
$X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{(x_2 a + x_1 x_2 b)} = g^{x_2 a} (\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_2 (x_3' + x_4')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{(x_4 a + x_3 x_4 b)} = g^{x_4 a} (\frac{\beta^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_4 (x_1' + x_2')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
Return $b = b'$;

FIGURE A.18: Game 7-8.1

Reduction for Game 7-8.m.1

Initialize (X, Y)
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $t \leq m$
  $X_2 \leftarrow X$;
If $t > m$
  $x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $t \leq m$
  $X_4 \leftarrow X$;
If $t > m$
  $x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $t \leq m$
  $\alpha \leftarrow X_2^{(x_1 + x'_3 + x'_4)\mathsf{pw}_{cs}}$;
If $t > m$
  $\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $t \leq m$
  $\beta \leftarrow X_4^{(x'_1 + x'_2 + x_3)\mathsf{pw}_{cs}}$;
If $t > m$
  $\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  If $t \leq m$
    $\mathsf{Rewrite\ alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
    If $g^c (X_1 X'_3 X'_4)^{d \cdot \mathsf{pw}_{cs}} \neq X_4^{\prime \mathsf{pw}_{cs}}$ then
      $\mathsf{Key} = X_2^a \left(\frac{\frac{1}{\alpha^{\mathsf{pw}_{cs}}}}{X_2^{(x'_3 + x'_4)}}\right)^b Y^{(c + d(x_1 + x'_3 + x'_4)\mathsf{pw}_{cs} - x'_4 \mathsf{pw}_{cs})}$;
    Else
      $\mathsf{Key} = X_2^a \left(\frac{\frac{1}{\alpha^{\mathsf{pw}_{cs}}}}{X_2^{(x'_3 + x'_4)}}\right)^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}}\right)^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else if $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  If $t \leq m$
    $\mathsf{Rewrite\ alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
    If $g^c (X'_1 X'_2 X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2^{\prime \mathsf{pw}_{cs}}$ then
      $\mathsf{Key} = X_4^a \left(\frac{\frac{1}{\beta^{\mathsf{pw}_{cs}}}}{X_4^{(x'_1 + x'_2)}}\right)^b Y^{(c + d(x'_1 + x'_2 + x_3)\mathsf{pw}_{cs} - x'_2 \mathsf{pw}_{cs})}$;
    Else
      $\mathsf{Key} = X_4^a \left(\frac{\frac{1}{\beta^{\mathsf{pw}_{cs}}}}{X_4^{(x'_1 + x'_2)}}\right)^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}}\right)^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else if $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi^i_S \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^j_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H(sid,Key,pw)
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}}}\right)^{x_2}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.K$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
If $b' = b$ return 1;
Else return 0; Abort.
Return $b = b'$

FIGURE A.19: Reduction for Game 7-8.m.1

Game 7-8.m.2.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $t \leq m$
  $\alpha \xleftarrow{\$} \mathcal{G}$;
If $t > m$
  $\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $t \leq m$
  $\beta \xleftarrow{\$} \mathcal{G}$;
If $t > m$
  $\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{x_2 a} (\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_2 (x_3' + x_4')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{x_4 a} (\frac{\beta^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_4 (x_1' + x_2')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
Return $b = b'$;

FIGURE A.20: Game 7-8.2

Game 9: Perfect forward secrecy.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; $T_6 \leftarrow \{\}$; $T_9 \leftarrow \{\}$;
bad_5=F; bad_9^1=F; bad_9^2=F;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$}$ NIZK.Backdoor( );
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, $(X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, $(X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_3' \notin$ List then:
  $x_3' \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X_3', \pi_3', S$);
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_4' \notin$ List then:
  $x_4' \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X_4', \pi_4', S$);
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\alpha \leftarrow g^w$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, $(X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, $(X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_1' \notin$ List then:
  $x_1' \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X_1', \pi_1', C$);
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_2' \notin$ List then:
  $x_2' \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X_2', \pi_2', C$);
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\beta \leftarrow g^w$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite alg$' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
  If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_2 a} (\frac{1}{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^j \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite alg$' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
  If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_4 a} (\frac{1}{\beta^{\frac{1}{\mathsf{pw}_{cs}}}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else if $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$
    If $(C, S) \notin$ Corr do $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
    If $(C, S) \in$ Corr $\wedge$ pw $= \mathsf{pw}_{cs} \wedge bad_9^2 =$ F
      then $T_9 \leftarrow T_9 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg})$;
    Rewrite alg $= [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} = X_2'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg})$;
    Rewrite alg $= [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} = X_4'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow$ Reveal(U, i); $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then bad_5=T;
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_9$ do bad_9^1=T;
If bad_9^2=T then bad_9^1=F;
Return $b = b'$;

FIGURE A.21: Game 9

Reduction for Game 9

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$;
$T_6 \leftarrow \{\}$; $T_{\mathsf{bad}_9} \leftarrow \{\}$; $T_{\mathsf{bad}_5} \leftarrow \{\}$;
bad_9^2=F; $(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $r = l$
  $X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
Else
  $x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 = g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $r = l$
  $X_4 \leftarrow X$; $x_4 \leftarrow \perp$;
Else
  $x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\alpha \leftarrow g^w$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\beta \leftarrow g^w$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
  If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_2 a} (\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_2 (x_3' + x_4')}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  If $x_2 \neq \perp$
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}}})^{x_2}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
  If $x_2 = \perp$
    $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
  If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_4 a} (\frac{\beta^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_4 (x_1' + x_2')}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  If $x_4 \neq \perp$
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}}})^{x_4}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
  If $x_4 = \perp$
    $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $x_2 = \perp \vee x_4 = \perp$
    If $(C, S) \notin \mathsf{Corr}$ do $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
    If $(C, S) \in \mathsf{Corr} \wedge \mathsf{pw} = \mathsf{pw}_{cs} \wedge bad_9^2 = \mathsf{F}$
      do $T_{\mathsf{bad}_9} \leftarrow T_{\mathsf{bad}_9} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg})$;
    Rewrite $\mathsf{alg} = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} = X_2'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    Rewrite $\mathsf{alg} = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} = X_4'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_{\mathsf{bad}_9}$ do $\mathcal{B}_9.\mathsf{Finalize}(g^{x^2})$;
Return $b = b'$;

FIGURE A.22: Reduction for Game 9

Abbreviations

AC Access Control 56
AGM Algebraic Group Model 16
AKE Authenticated Key-Exchange Protocol 13
alg-SE-NIZK Algebraic Simulation Sound Extractable Non-Interactive Zero-Knowledge Proof 22

CDH Computational Diffie-Hellman 63
CHL Credential-hiding Login 32
CRS Common Reference String 16
CSqDH Computational Square Diffie-Hellman 90
CTGDH Computational Triple Group Diffie-Hellman 89

DDH Decisional Diffie-Hellman 12
DH-KE Diffie-Hellman Key-Exchange 2
DHP Diffie-Hellman Problem 12
DLP Discrete Log Problem 12
DPP Device Provisioning Protocol 7

FtG Find then Guess 39

GGM Generic Group Model 16

HC Honey Checker ix

KDC Key-Distribution Center 2

NIZK Non-Interactive Zero-Knowledge 18

PAKE Password authenticated key exchange iii
PFS Perfect Forward Secrecy 34
PKI Public-Key Infrastructure iii
PPT Probabilistic Polynomial Time 15

RO Random Oracle 16
RoR Real or Random 39

SAS-MA Short-Authenticated-String Message Authentication 32
SE-NIZK Simulation Sound Extractable Non-Interactive Zero-Knowledge Proof 22
SSL Secure Sockets Layer 5

TFA Two-Factor Authentication 32

UC Universally Composable 34

wFS Weak Forward secrecy 34

Bibliography

[1Pa18] 1Password Security Design. https://1password.com/files/1Password, 27-02-2018.

[AB19] M. Abdalla and M. Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. https://eprint.iacr.org/2019/1194.

[ABB+20] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. Cryptology ePrint Archive, Report 2020/320, 2020. https://eprint.iacr.org/2020/320.

[ABM15] M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*, pages 571–587. IEEE Computer Society Press, May 2015.

[ABR+21] M. Abdalla, M. Barbosa, P. B. Rønne, P. Y. Ryan, and P. Šala. Security characterization of j-pake and its variants. Cryptology ePrint Archive, Report 2021/824, 2021. https://ia.cr/2021/824.

[AFP05] M. Abdalla, P. Fouque, and D. Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In *Public-Key Cryptography – PKC 2005*, *LNCS* 3386, pages 65–84. Springer, 2005.

[AHH21] M. Abdalla, B. Haase, and J. Hesse. Security analysis of cpace. Cryptology ePrint Archive, Report 2021/114, 2021. https://eprint.iacr.org/2021/114.

[Ama20] Amazon, Inc. Amazon Cognito: User Authentication Flow, April 2020. https://aws.amazon.com/cognito/, as of December 14, 2021.

[AP05] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, *LNCS* 3376, pages 191–208. Springer, Heidelberg, February 2005.

[BBBB10] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *Computer Security – ESORICS 2010*, pages 286–302, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BBCW20] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi. Provable security analysis of fido2. Cryptology ePrint Archive, Report 2020/756, 2020. https://eprint.iacr.org/2020/756.

[BCJ+19] T. Bradley, J. Camenisch, S. Jarecki, A. Lehmann, G. Neven, and J. Xu. Password-authenticated public-key encryption. Cryptology ePrint Archive, Report 2019/199, 2019. https://eprint.iacr.org/2019/199.

[BDZ03] F. Bao, R. Deng, and H. Zhu. Variations of diffie-hellman problem. In *ICICS 2003*, *LNCS* 2836, pages 301–312. Springer, 2003.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing - STOC 1988*, pages 103–112. ACM, 1988.

[BHvOS12] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy, SP 2012*, pages 553–567. IEEE Computer Society, 2012.

[BHvOS15] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. Passwords and the evolution of imperfect authentication. *Commun. ACM*, 58(7):78–87, June 2015.

[BIO+17] J. Becerra, V. Iovino, D. Ostrev, P. Šala, and M. Škrobot. Tightly-secure pak(e). Cryptology ePrint Archive, Report 2017/1045, 2017. `https://ia.cr/2017/1045`.

[BJLS16] C. Bader, T. Jager, Y. Li, and S. Schäge. On the impossibility of tight cryptographic reductions. In *Advances in Cryptology – EUROCRYPT 2016*, pages 273–304, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BK09] J. Bender and D. Kuegler. Introducing the pace solution. `https://bit.ly/3iybFFN/`, 25-02-2009.

[BM92] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *1992 IEEE Symposium on Research in Security and Privacy, SP 1992*, pages 72–84, 1992.

[BM97] S. Blake-Wilson and A. Menezes. Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques. In *Security Protocols, 5th International Workshop*, *LNCS* 1361, pages 137–158. Springer, 1997.

[BMP00] V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *EUROCRYPT 2000*, *LNCS* 1807, pages 156–171. Springer, Heidelberg, May 2000.

[BOM+19] M. Beunardeau, F.-E. E. Orche, D. Maimut, D. Naccache, P. B. Roenne, and P. Y. Ryan. Authenticated key distribution: When the coupon collector is your enemy. Cryptology ePrint Archive, Report 2019/1499, 2019. `https://ia.cr/2019/1499`.

[Bon12] J. Bonneau. *Guessing human-chosen secrets*. PhD thesis, University of Cambridge, UK, 2012.

[BOS19] J. Becerra, D. Ostrev, and M. Skrobot. Forward secrecy of spake2. Cryptology ePrint Archive, Report 2019/351, 2019. `https://eprint.iacr.org/2019/351`.

[Boy09] X. Boyen. Hidden credential retrieval from a reusable password. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 228–238, New York, NY, USA, 2009. ACM.

[BPR00] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, *LNCS* 1807, pages 139–155. Springer, Heidelberg, May 2000.

[BR93] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology — CRYPTO 1993*, *LNCS* 773, pages 232–249. Springer, 1993.

[BR94] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, *LNCS* 773, pages 232–249. Springer, Heidelberg, August 1994.

[BR95] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: the three- party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, STOC '95*, pages 57–66. ACM, 1995.

[BR06] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006*, pages 409–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Bre19] J. M. L. Brecerra. *Provable Security Analysis for the Password Authenticated Key Exchange Problem.* PhD thesis, University of Luxembourg, 2019.

[BRRS18] J. Becerra, P. B. RØnne, P. Y. A. Ryan, and P. Sala. Honeypakes. In *Security Protocols XXVI - 26th International Workshop, Cambridge, UK, March 19-21, 2018, Revised Selected Papers*, *Lecture Notes in Computer Science* 11286, pages 63–77. Springer, 2018.

[BŠŠ17] J. Becerra, P. Šala, and M. Škrobot. An Offline Dictionary Attack against zkPAKE Protocol. Cryptology ePrint Archive, Report 2017/961, 2017. https://eprint.iacr.org/2017/961.

[Can01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[CGCG+19] K. Cohn-Gordon, C. Cremers, K. Gjøsteen, H. Jacobsen, and T. Jager. Highly efficient key exchange protocols with optimal tightness – enabling real-world deployments with theoretically sound parameters. Cryptology ePrint Archive, Report 2019/737, 2019. `https://ia.cr/2019/737`.

[CH14] D. Clarke and F. Hao. Cryptanalysis of the Dragonfly Key Exchange Protocol. *IET Information Security*, 8(6):283–289, 2014.

[CHK+05] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally Composable Password-Based Key Exchange. In *Advances in Cryptology – EUROCRYPT 2005*, *LNCS* 3494, pages 404–421. Springer, 2005.

[CK01] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT 2001*, *LNCS* 2045, pages 453–474. Springer, Heidelberg, May 2001.

[Den11] F. Dennis. What you need to know about the diginotar hack. `https://bit.ly/3fSGeV9`, 2011.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DR08] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. `http://www.rfc-editor.org/rfc/rfc5246.txt`.

[EKSS09] J. Engler, C. Karlof, E. Shi, and D. Song. Is it too late for PAKE? In *Web 2.0 Security and Privacy Workshop 2009 (W2SP 2009)*, May 2009.

[Fac19] Server-compromise facebook. `https://reut.rs/35vPQ2v`, 25-02-2019.

[FK00] W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '00, pages 176–180, Washington, DC, USA, 2000. IEEE Computer Society.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 33–62. Springer, Heidelberg, August 2018.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[GEAR09] R. Giot, M. El-Abed, and C. Rosenberger. Greyc keystroke: a benchmark for keystroke dynamics biometric systems. In *2009 IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*. IEEE, 2009.

[GJK21] Y. Gu, S. Jarecki, and H. Krawczyk. Khape: Asymmetric pake from key-hiding key exchange. Cryptology ePrint Archive, Report 2021/873, 2021. `https://ia.cr/2021/873`.

[GLRS17] Z. A. Genc, G. Lenzini, P. Y. A. Ryan, and I. V. Sandoval. A security analysis, and a fix, of a code-corrupted honeywords system. 2017.

[GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2)/ 270-299, 1984.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[Gro06] J. Groth. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In *Advances in Cryptology - ASIACRYPT 2006*, *LNCS* 4284, pages 444–459. Springer, 2006.

[Hao21] F. Hao. Prudent practices in security standardization. Cryptology ePrint Archive, Report 2021/839, 2021. `https://eprint.iacr.org/2021/839`.

[Har15] D. Harkins. Dragonfly Key Exchange. RFC 7664, RFC Editor, November 2015.

[Har18] D. Harkins. Public Key Exchange. Internet-Draft draft-harkins-pkex-06, Internet Engineering Task Force, August 2018. Work in Progress.

[Has19] T. Hashcat. hashcat - advanced password recovery. `https://hashcat.net/hashcat/`, 25-02-2019.

[HL18] B. Haase and B. Labrique. Aucpace: Efficient verifier-based pake protocol tailored for the iiot. Cryptology ePrint Archive, Report 2018/286, 2018. `https://ia.cr/2018/286`.

[HL19] B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/7384`.

[HR10] F. Hao and P. Ryan. J-PAKE: Authenticated Key Exchange without PKI. *Transactions on Computational Science*, 11:192–206, 2010.

[HS14] F. Hao and S. Shahandashti. The speke protocol revisited. 12 2014.

[IEE02] Standard Specifications for Password-Based Public Key Cryptographic Techniques. Standard, IEEE Standards Association, Piscataway, NJ, USA, 2002.

[IET] Internet Engineering Task Force. `https://www.ietf.org/`.

[ISO09] ISO/IEC 11770-4:2006/cor 1:2009, Information Technology – Security techniques – Key Management – Part 4: Mechanisms Based on Weak Secrets. Standard, International Organization for Standardization, Genève, Switzerland, 2009.

[Jab96] D. P. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[Jab97] D. P. Jablon. Extended password key exchange protocols immune to dictionary attack, 1997.

[JJK+21] S. Jarecki, M. Jubur, H. Krawczyk, N. Saxena, and M. Shirvanian. Two-factor password-authenticated key exchange with end-to-end security. *ACM Trans. Priv. Secur.*, 24(3):17:1–17:37, 2021.

[JKSS18] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Two-factor authentication with end-to-end password security. In *Public-Key Cryptography – PKC 2018*, pages 431–461, Cham, 2018. Springer International Publishing.

[JKX18] S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks. In *Advances in Cryptology – EUROCRYPT 2018*, LNCS. Springer, 2018.

[joh19] John the ripper password cracker. `https://www.openwall.com/john/`, 25-02-2019.

[JR13] A. Juels and R. L. Rivest. Honeywords: making password-cracking detectable. In *ACM CCS 2013*, pages 145–160. ACM Press, November 2013.

[JY98] M. Joye and S. Yen. Id-based secret-key cryptography. *ACM SIGOPS Oper. Syst. Rev.*, 32(4):33–39, 1998.

[KL07] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KM13] F. Kiefer and M. Manulis. Oblivious pake: Efficient handling of password trials. Cryptology ePrint Archive, Report 2013/127, 2013. `https://ia.cr/2013/127`.

[KOY01] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, *LNCS* 2045, pages 475–494. Springer, Heidelberg, May 2001.

[Kra03] 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, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Kra05] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO 2005*, *LNCS* 3621, pages 546–566. Springer, Heidelberg, August 2005.

[Lin12] Server-compromise linkedin. `https://www.linkedin.com/help/linkedin/answer/69603/notice-of-data-breach-may-2016`, 25-02-2012.

[LMR20] K. Lewi, P. Mohassel, and A. Roy. Single-message credential-hiding login. Cryptology ePrint Archive, Report 2020/1509, 2020. `https://eprint.iacr.org/2020/1509`.

[LST16] J. Lancrenon, M. Skrobot, and Q. Tang. Two more efficient variants of the J-PAKE protocol. In *ACNS 16*, *LNCS* 9696, pages 58–76. Springer, Heidelberg, June 2016.

[Lv15] J. Lancrenon and M. Škrobot. On the Provable Security of the Dragonfly Protocol. In *International Conference on Information Security*, pages 244–261. Springer, 2015.

[Mac01] 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.

[Mac02] P. MacKenzie. The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46, 2002.

[Mag16] Magic Wormhole. `https://github.com/warner/magic-wormhole`, 2016.

[Mer82] M. Merritt. Key reconstruction. In *CRYPTO'82*, pages 321–322. Plenum Press, New York, USA, 1982.

[MPS00] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on rsa. In *Advances in Cryptology — ASIACRYPT 2000*, pages 599–613, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[MRA15] K. Mochetti, A. C. D. Resende, and D. F. Aranha. zkpake : A simple augmented pake protocol. 2015.

[MSKD16] M. Manulis, D. Stebila, F. Kiefer, and N. Denham. Secure Modular Password Authentication for the Web Using Channel Bindings. *International Journal of Information Security*, 15(6):597–620, 2016.

[MTI86] T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.

[MTT18] T. Mizuide, A. Takayasu, and T. Takagi. Tight reductions for diffie-hellman variants in the algebraic group model. Cryptology ePrint Archive, Report 2018/1220, 2018. `https://ia.cr/2018/1220`.

[NYHR05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). RFC 4120, RFC Editor, July 2005. `http://www.rfc-editor.org/rfc/rfc4120.txt`.

[Omn20] Omnicom. Social media benchmark report. `https://www.omnicoreagency.com/social-media-statistics/`, 2020.

[Ope16] OpenSSL. `https://www.openssl.org/`, April 2, 2016.

[OWT09] Y. Oiwa, H. Watanabe, and H. Takagi. Pake-based mutual HTTP authentication for preventing phishing attacks. *CoRR*, abs/0911.5230, 2009.

[Pal16] Pale Moon. `http://www.palemoon.org`, April 2, 2016.

[Pax21] Server-compromise paxful. `https://bit.ly/3iQAH3x`, 25-02-2021.

[Poi12] D. Pointcheval. Password-Based Authenticated Key Exchange. In *Public Key Cryptography - PKC 2012*, *LNCS* 7293, pages 390–397. Springer, 2012.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology - EUROCRYPT 1996*, *LNCS* 1070, pages 387–398. Springer, 1996.

[PS20] C. Patton and T. Shrimpton. Quantifying the security cost of migrating protocols to practice. Cryptology ePrint Archive, Report 2020/573, 2020. `https://ia.cr/2020/573`.

[PV05] P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *ASIACRYPT 2005*, *LNCS* 3788, pages 1–20. Springer, Heidelberg, December 2005.

[PW17] D. Pointcheval and G. Wang. Vtbpeke: Verifier-based two-basis password exponential key exchange. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 301–312, New York, NY, USA, 2017. ACM.

[Rya17] P. Y. A. Ryan. Auditable pakes: Approaching fair exchange without a ttp (transcript of discussion). In *Security Protocols XXV*, pages 298–305, Cham, 2017. Springer International Publishing.

[SALR21] I. V. Sandoval, A. Atashpendar, G. Lenzini, and P. Y. A. Ryan. Pake-mail: authentication and key management in decentralized secure email and messaging via PAKE. *CoRR*, abs/2107.06090, 2021.

[Sch90] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, *LNCS* 435, pages 239–252. Springer, Heidelberg, August 1990.

[scr] scrypt-parameters. `https://blog.filippo.io/the-scrypt-parameters/`.

[Sho99] V. Shoup. On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012, 1999. `http://eprint.iacr.org/1999/012`.

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[Sho20] V. Shoup. Security analysis of spake2+. Cryptology ePrint Archive, Report 2020/313, 2020. `https://eprint.iacr.org/2020/313`.

[SJ15] M. Stapelberg and S. Josefsson. Universal 2nd Factor (U2F) Authentication for Secure Shell (SSH). Internet-Draft draft-josefsson-secsh-u2f-00, Internet Engineering Task Force, February 2015. Work in Progress.

[SJKS17] M. Shirvanian, S. Jarecki, H. Krawczyk, and N. Saxena. SPHINX: A password store that perfectly hides passwords from itself. In *ICDCS*, pages 1094–1104. IEEE Computer Society, 2017.

[SKFB21] N. Sullivan, D. H. Krawczyk, O. Friel, and R. Barnes. OPAQUE with TLS 1.3. Internet-Draft draft-sullivan-tls-opaque-01, Internet Engineering Task Force, February 2021. Work in Progress.

[Skr17] M. Skrobot. *On the Composability and Security of Game based Password Authenticated Key Exchange*. PhD thesis, University of Luxembourg, 2017.

[SL18] M. Skrobot and J. Lancrenon. On composability of game-based password authenticated key exchange. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 443–457, 2018.

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, CCS '96, page 31–37, New York, NY, USA, 1996. Association for Computing Machinery.

[Szy06] M. Szydlo. A note on chosen-basis decisional diffie-hellman assumptions. In *Financial Cryptography and Data Security*, pages 166–170, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Thr12] Threema: The messanger that puts the security and privacy first. `https://threema.ch/en`, 2012.

[Thr16] Thread Protocol. `http://threadgroup.org/`, April 2, 2016.

[TWMP07] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the secure remote password (srp) protocol for tls authentication. RFC 5054, RFC Editor, November 2007.

[Vau05] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology – CRYPTO 2005*, pages 309–326, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[VR19] M. Vanhoef and E. Ronen. Dragonblood: Analyzing the dragonfly handshake of wpa3 and eap-pwd. Cryptology ePrint Archive, Report 2019/383, 2019. `https://ia.cr/2019/383`.

[Wif18] Wifi easy to connect. `https://www.wi-fi.org/discover-wi-fi/wi-fi-easy-connect/`, 2018.

[Wu98] T. D. Wu. The secure remote password protocol. In *NDSS'98*. The Internet Society, March 1998.

[YBAG04] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5):25–31, September 2004.

[Yub] Yubikey. `https://www.yubico.com/`.

RÉSUMÉ

La grande majorité des communications sur Internet et sur les réseaux privés repose fortement sur des infrastructures à clé publique (PKI). Une solution possible, pour réduire la complexité induite par les PKIs, consiste à utiliser des protocoles d'échange de clés authentifiés par mots de passe (PAKE). Les protocoles PAKE permettent une communication sécurisée entre deux parties qui ne partagent qu'un secret de faible entropie (mot de passe). Les PAKEs furent introduits dans les années 1990. Les premiers modèles et preuves de sécurité ont suivi au début des années 2000. Ainsi, il devint clair que les PAKEs ont un potentiel de déploiement à grande échelle - comblant le vide, là où l'infrastructure à clé publique est insuffisante. Le fait que les PAKEs permettent de se passer d'un PKI, leur résistance aux attaques de hameçonnage et la confidentialité qu'ils offrent ne sont que quelques-unes des propriétés rendant les PAKEs intéressants à étudier. Cette thèse comporte trois nouveaux résultats concernant divers aspects des PAKEs : une attaque sur une proposition PAKE existante, une application de PAKEs permettant la détection de fuites de mots de passe (HoneyPAKEs), et une analyse de sécurité du protocole J-PAKE qui est utilisé dans la pratique. Cette dernière analyse s'appliquant également aux variantes de J-PAKE. Dans notre premier travail, nous proposons une analyse empirique du protocole zkPAKE proposé en 2015. Nos résultats démontrent que zkPAKE n'est pas sûr contre les attaques par dictionnaire hors ligne, qui est l'une des exigences de sécurité de base des protocoles PAKE. De plus, nous exhibons une implémentation d'une attaque par dictionnaire hors ligne efficace qui souligne que lors de la proposition d'un nouveau protocole, qu'il est nécessaire fournir une preuve de sécurité rigoureuse. Notre seconde propose un mécanisme de sécurité combiné appelé HoneyPAKE. La construction HoneyPAKE vise à détecter la perte de fichiers de mots de passe et garantit que le PAKE utilisé protège intrinsèquement les mots de passe. Cela rend la partie PAKE du HoneyPAKE plus résistante aux compromissions de serveurs et aux attaques par pré-calcul. Ce faisant, nous ajoutons une garantie de sécurité sérieuse contre les menaces sur les communicationsclient-serveur. Notre troisième contribution facilite l'adoption plus large des PAKE. Dans ce travail, nous revisitons J-PAKE, en le simplifiant. Cette simplification s'effectue en supprimant une preuve à divulgation nulle non interactive du dernier tour du protocole, résultant ainsi en une version plus légère appelée sJ-PAKE. De plus, nous prouvons que sJ-PAKE est sûr dans le modèle basé sur le jeu de l'indiscernabilité dit Réel-ou-Aléatoire (Real-or-Random), satisfaisant ainsi également la notion de secret avançant (Forward Secrecy).

MOTS CLÉS

Authentification, preuve, clé public, échange de clés, le mot de passe, attaques

ABSTRACT

The vast majority of communication on the Internet and private networks heavily relies on Public-key infrastructure (PKI). One possible solution, to avoid complexities around PKI, is to use Password Authenticated Key-Exchange (PAKE) protocols.
PAKE protocols enable a secure communication link between the two parties who only share a low-entropy secret (password). PAKEs were introduced in the 1990s, and with the introduction of the first security models and security proofs in the early 2000s, it was clear that PAKEs have a potential for wide deployment - filling the gap where PKI falls short. PAKEs' PKI-free nature, resistance to phishing attacks and forward secrecy are just some of the properties that make them interesting and important to study. This dissertation includes three works on various aspects of PAKEs: an attack on an existing PAKE proposal, an application of PAKEs in login (for password leak detection) and authentication protocols (HoneyPAKEs), and a security analysis of J-PAKE protocol that is used in practice, and its variants. In our first work, we provide an empirical analysis on zkPAKE protocol proposed in 2015. Our findings show that zkPAKE is not safe against offline dictionary attacks, which is one of the basic security requirements of the PAKE protocols. Further, we demonstrate an implementation of an efficient offline dictionary attack, which emphasizes, when proposing a new protocol, it is necessary to provide a rigorous security proof. In our second contribution, we propose a combined security mechanism called HoneyPAKE. The HoneyPAKE construction aims to detect the loss of password files and ensures that PAKE intrinsically protects that password. This makes the PAKE part of the HoneyPAKE more resilient to server-compromise and pre-computation attacks that are a serious security threat in a client-server communication. Our third contribution facilitates the wider adoption of PAKEs. In this work, we revisit J-PAKE, simplify it by removing a non-interactive zero knowledge proof from the last round of the protocol and derive a lighter and more efficient version called sJ-PAKE. Furthermore, we prove sJ-PAKE secure in the indistinguishability game-based model, the so-called Real-or-Random, also satisfying the notion of perfect forward secrecy.

KEYWORDS

Authentication, proofs, public key, key-exchange, password, attacks