PhD-FSTC-2017-09
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 13/01/2017 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Marjan ŠKROBOT
Born on 25 November 1987 in Sremska Mitrovica (Serbia)

ON COMPOSABILITY AND SECURITY OF GAME-BASED PASSWORD-AUTHENTICATED KEY EXCHANGE

Dissertation defence committee

Dr Peter Y.A. Ryan, dissertation supervisor
Professor, Université du Luxembourg

Dr Cas Cremers
Professor, University of Oxford

Dr Sjouke Mauw, Chairman
Professor, Université du Luxembourg

Dr Jintai Ding, Vice Chairman
Professor, University of Cincinnati

Dr Jean Lancrenon
Itrust consulting sàrl

Abstract

The main purpose of Password-Authenticated Key Exchange (PAKE) is to allow secure authenticated communication over insecure networks between two or more parties who only share a low-entropy password. It is common practice that the secret key derived from a PAKE execution is used to authenticate and encrypt some data payload using symmetric key protocols. Unfortunately, most PAKEs of practical interest, including three protocols considered in this thesis, are studied using so-called *game-based* models, which – unlike simulation models – do not guarantee secure composition *per se*. However, Brzuska et al. (CCS 2011) have shown that a middle ground is possible in the case of authenticated key exchange that relies on *Public-Key Infrastructure* (PKI): the game-based models do provide secure composition guarantees when the class of higher-level applications is restricted to symmetric-key protocols. The question that we pose in this thesis is whether or not a similar result can be exhibited for PAKE. Our work answers this question positively. More specifically, we show that PAKE protocols secure according to the game-based Real-or-Random (RoR) definition of Abdalla et al. (PKC 2005) allow for automatic, secure composition with arbitrary, higher-level symmetric key protocols. Since there is evidence that most PAKEs secure in the Find-then-Guess (FtG) model of Bellare et al. (EUROCRYPT 2000) are in fact secure according to the RoR definition, we can conclude that nearly all provably secure PAKEs enjoy a certain degree of composition, one that at least covers the case of implementing secure channels.

Although many different protocols that accomplish PAKE have been proposed over last two decades, only a few newcomers managed to find their way to real world applications - albeit lacking an intense and prolonged public scrutiny. As a step in the direction of providing one, this dissertation considers the security and efficiency of two relatively recently proposed PAKE protocols - Dragonfly and J-PAKE. In particular, we prove the security of a very close variant of Dragonfly employing the standard FtG model which incorporates forward secrecy. Thus, our work confirms that Dragonfly's main flows are sound. Furthermore, we contribute to the discussion by proposing and

examining (in the RoR model of security) two variants of J-PAKE - which we call RO-J-PAKE and CRS-J-PAKE - that each makes the use of two less zero-knowledge proofs than the original protocol, at the cost of an additional security assumption. Our work reveals that CRS-J-PAKE has an edge in terms of efficiency over J-PAKE for both standard group choices: subgroups of finite fields and elliptic curves. The same is true for RO-J-PAKE, but only when instantiated with elliptic curves.

Acknowledgements

First and foremost, I would like to express gratitude to my thesis mentor, Prof. Peter Y.A. Ryan, for the opportunity he gave me to carry out my Ph.D. research at the APSIA group, as well as for his guidance and support throughout this study. As my advisor, he has suggested fruitful research directions for me to pursue and later gave me the freedom to choose my research topics. I am very appreciative for all he has done for me.

As my daily advisor, dr. Jean Lancrenon has patiently and skillfully guided me through (at the time) very foreign world of public key cryptography and provable security. Although fresh Ph.D. graduate himself, Jean has tremendously contributed to my development as a researcher and I feel especially indebted to him. I would like to thank Jean (and Peter) for the many enjoyable discussions we had on the topic of key exchange protocols. Jean has truly been my closest research collaborator.

I would also like to thank Prof. Cas Cremers for serving as a member of my thesis committee and later being the member of my jury. His comments and questions were highly beneficial (particularly in the beginning of my Ph.D. program) and helped me put things in a wider perspective.

Besides Peter, Jean, and Cas, I would like to express my gratitude to the other members of my jury, Prof. Jintai Ding and Prof. Sjouke Mauw, for reading my manuscript, attending my public defence, and providing valuable feedback.

I also wish to acknowledge my co-authors, dr. Jean Lancrenon and dr. Qiang Tang, for their help and contribution.

A word of appreciation goes to the University of Luxembourg and the SnT – Interdisciplinary Centre for Security, Reliability, and Trust – for providing a great environment for young researchers and allowing their academic and personal growth. I am also very grateful for the financial support given to me by the Doctoral School of Computer Sciences and Computer Engineering. I thank Catherine Violet for her help in orchestrating this support.

My warm gratitude goes to my office-mates, colleagues and friends from APSIA and SnT: Masoud, Afonso, Massimo, Miguel, Rosario, Jean, Peter, Qiang, Martin, Phu, Walter, Yu Li, Jose, Jorge, Dayana, Jean-Louis, Gabriele, Jun, Arash, Balazs, Vincenzo, Alfredo, Dimiter, Wojtek, Dalia, Itzel, Ziya, Samir, Christian, Gergei and Peter Browne. I feel greatly privileged to have been given the opportunity to work and study together with such a diverse group of people that are generous in sharing their skills and knowledge. I have enjoyed our stimulating discussions.

I would also like to use this opportunity to thank my master thesis advisors, dr. Elena Andreeva and dr. Bart Mennink, for helping me and encouraging me to pursue my Ph.D. studies.

I deeply thank my parents, Ruža and Pavle, and my brothers Ivan and Dragutin, for their unconditional support and endless love that will always keep me going. Here I would like to extend my sincerest thanks for the rest of my family and friends without whom all of this would not have been possible.

Finally, I would like to thank my wife Ljiljana for her constant support, understanding, and love. She has been my best friend and great companion, continually reminding me what is truly important in life and helping me to get through this Ph.D. journey. My life, including this text, would not be the same without you.

Thank you all once more.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The problem of finding a robust solution for secure human authentication in the digital environment, although not fundamental as the P versus NP problem – whose solution would have far-reaching implications in cryptography – is almost as old, originating since the early 1960s when the first computer system to deploy passwords, The Compatible Time-Sharing System (CTSS)[1], was developed at MIT [103]. Today, with the increasing number of Internet-enabled devices and third party services that require authentication, the need for a secure, general-purpose, and simple-to-use authentication mechanism is of great importance. One can only imagine the potential risk posed by the Internet of Things (IoT), which will soon become a reality, where almost any object of human use, whether at home, in the office, in a public place or within an environment in the wider sense will be internet-reachable and potentially compromised.

The prevalence and perseverance of text passwords and pins as authentication means of choice for humans today, despite many well-studied limitations and security problems [93], has been mind-boggling for security researchers. During the last fifty years, a significant number of alternatives to passwords has been proposed and analyzed [19]. However, none of the alternative solutions has managed to eliminate passwords from our lives. The reason for this, according to the comprehensive comparison study on different web authentication schemes done by Bonneau et al. [19], is the lack of deployability of proposed alternatives in comparison to passwords. Namely, accessibility, negligible cost-per-user, server and browser compatibility, and maturity

[1] In this system, a password for each account was stored in an unencrypted CTSS password file. The same system was a victim of a software bug which caused the password file to be printed on the terminal upon login instead of a daily message [93].

of existing solutions are identified as factors that ease the deployment and most probably contribute to the resilience of passwords in our computer systems. Thus, despite all the limitations of passwords and the existence of user-friendly alternatives with better security guarantees such as Pico [113] or OpenID [99], one thing is clear: passwords are here to stay in the foreseeable future [62]. This fact places cryptographic schemes that can advance the current state of affairs in password authentication - such as *Password-Authenticated Key Exchange* - in the spotlight.

1.2 Password-based Authentication Protocols

Here we will briefly introduce authentication protocols and password-authenticated key exchange. Then we will present potential attacks against which we want to defend authentication protocols. After this, we will see how password authentication has evolved during the last forty years. For more details on password security research, we refer the reader to [18, 74].

1.2.1 Authentication Protocols

As in a typical human-computer authentication scenario, we will be considering two-party authentication protocols in which a *user* or *client* has to prove its identity to a skeptical *verifier* (typically a *server*) by submitting the correct *credential*. It is assumed that the two entities share a common secret apriori. In real-world applications such as authentication over the internet, users may have a software (*browser*) operating on their behalf, and the union of these two will be modelled as a client.

1.2.2 Password-Authenticated Key Exchange

Authenticated Key Exchange (AKE) is a cryptographic service run between two or more parties over a network with the aim of agreeing on a secret, high-quality, session key to use in higher-level applications (e.g., to create an efficient and secure channel.) One talks of *Password*-Authenticated Key Exchange (PAKE) if the message flows of the protocol itself are authenticated by means of a *low-entropy secret* held by each user. Throughout our work, we will refer to the term low-entropy secret – which may include passwords, pins, and passphrases – simply as a *password*. Also, while password-based protocols in general may assume the existence of *Public-Key-Infrastructure* (PKI), in our work we will deal exclusively with password-only protocols. In contrast to generic authentication protocols, an additional consequence of a PAKE execution is the establishment of a cryptographically strong session key.

PAKE vs PKI-AKE. There are several practical advantages of using the password-only authenticated key exchange protocols rather than PKI-based alternatives. It is known to be very difficult to implement and maintain a functional PKI. Problems that may arise in the PKI setting include user registration, misbehaving Certification Authorities (CAs), certificate revocation issues, improper management of the keys, certificate validation issues and poor usability, susceptibility to phishing attacks, etc.

On the contrary, PAKE maintenance seems to be simpler and more flexible: In the initial phase, protocol participants should exchange passwords in a secure manner and fix public parameters that are known to all participants (including adversary). Therefore, in PAKE setting user registration is straightforward, there is no need for key revocation, the process can be made user-intuitive, phishing attacks are diminished, and participants are free of storing any secrets[2].

1.2.3 Possible Attacks

An adversary's goal in authentication protocols is to acquire one or more valid username and password pairs (credentials), and there exist several known paths to achieve this: either by directly targeting the protocol by guessing valid credentials, or by circumventing the security mechanism altogether.

Guessing Attacks. The inherent danger in the setup where passwords are used for authentication is its vulnerability to *dictionary attacks*. Depending on the adversarial strategy one can distinguish between two types of attacks: *online* and *offline* dictionary attacks.

An *online dictionary attack* consists of repeatedly sending candidate credentials to the verifier in order to test their validity. These attacks can be successfully curbed with a well-designed password protection policy that would limit the number of guesses which can be made in a given time frame.

In an *offline dictionary attack*, the adversary acts in two stages: in the first (online) stage, the adversary tries to collect – either by eavesdropping or impersonating a user – some function of credentials which will serve as the password verifier. Then, in the second (offline) stage, the adversary will try to correlate the acquired verifier with password guesses to determine the correct password. Notice that in contrast to a typical *brute force* attack, in which a secret is assumed to be uniformly distributed over

[2]This claim is valid for certain practical scenarios, such as Magic Wormhole [114] in which both (human) users input a short pin to run PAKE. On the other hand, in systems in which the server stores the password file, this claim is false.

the key space and searched systematically, in the second stage of the offline dictionary attack the adversary usually chooses to traverse those possibilities which are deemed most likely to succeed. This type of offline guessing is called *password cracking* and is inhibited in case secure PAKE is used as the authentication mechanism.

Other Types of Attacks. One of the most successful types of attacks that circumvent secure authentication are (spear) *phishing* attacks. In an attempt to obtain secret credentials, the adversary tries to disguise himself as a valid server when honest users try to authenticate to the service. Moore and Clayton [92] estimate that between 280 000 and 560 000 people are tricked each year into disclosing their credentials to phishing websites. This is very unfortunate, especially due to the fact that the appropriate use of PAKEs would completely eradicate such attacks.

A simple attack that also very often works is the *password sniffing attack*. This is an essentially an *eavesdropping* attack in which the adversary observes credentials during their transmission, which is not encrypted. The fact that today, according to [82], only 15.4% of the top 10 000 websites are using SSL by Default is quite surprising. This offers a good indication that a lot of secret credentials are transmitted through the network in an unencrypted form. PAKEs could be used to prevent such attacks.

Another way of stealing credentials is by running a malicious software on the target's device, whose role is to register credentials at the time of their entry. Such attacks are known as *key-logging* attacks; they are out of the scope of this thesis.

1.2.4 Evolution of Password Authentication

In all password-based authentication protocols that we consider, it is assumed that two peers share a password in advance and use it as the basis of authentication. Over the years password-based authentication protocols have evolved rather slowly considering their wide use and importance. Here we will briefly cover their evolution.

Passwords in Clear. Most of the password authentication methods used during the 1980s and the first half of 1990s included sending passwords in clear over the network. This is obviously completely insecure. Probably the most known example is the *basic access authentication* method used for a client authentication when making a connection request in *Hypertext Transfer Protocol* (HTTP) [43]. In this case, user credentials are only encoded with Base64 data encoding and then transmitted over the network without any encryption or hashing operation being involved. Another protocol in which plaintext passwords appear on the network is the *Password*

Authentication Protocol [81], known as PAP. This protocol is one of two possible authentication mechanisms specified in *Point-to-Point Protocol* (PPP)[3] [110].

Hash of Password. Another insecure approach that is often used today in many network protocols applies a hash function to the user's credentials (username and password) before sending them over the network. The hash function may also include a server and/or client created nonce as input. Two solutions that took this approach in solving password authentication are the *digest access authentication* mechanism and the *Challenge Handshake Authentication Protocol* (CHAP) [111]. The former is used in HTTP as a replacement for the completely insecure basic access authentication method, while the latter is a modest upgrade over PAP in PPP[4].

It should be noted that sometimes in practice a password is used for the authentication when high-entropy symmetric keys are expected (Pre-Shared Key (PSK) based setting). Some examples of protocols where misuse of passwords is possible include EAP-GPSK [33] and TLS-PSK [7]. These protocols, when used with a password instead of a cryptographically strong key for authentication, are susceptible to *dictionary* attacks.

Password over TLS. Probably the most prominent paradigm for password authentication that is deployed on the Internet today is the one for which Manulis et al. [89] coined the term *HTML-forms-over-TLS*. In this approach, a Transport Layer Security (TLS) [37] channel is first established, usually between the user's browser (or mobile application) and a server that offers some service (e.g. online banking, social networks, cloud services). Further, the server sends an HTML form over the secure channel (encrypted and server-authenticated) for the user to fill in with her password and username in order to authenticate herself. Although this approach can be considered secure [68, 77], there are several attacks that may circumvent security mechanisms and allow for the adversary to acquire credentials. Namely, in TLS, as part of server authentication, it is assumed that operational PKI is in place and that users will correctly validate the server's X.509 certificate before conveying their credentials. However, in practice, an adversary may trick a user to disclose her credentials to a fake but genuine-looking website, or Certification Authorities (CAs) may

[3]PPP is a data link protocol used to establish a simple bi-directional link which would allow transport of packets between two peers. This protocol is widely used in physical networks such as phone line, fiber optic links such as SONET, and even for a customer dial-up access to the internet.

[4]PAP and CHAP protocols are used in PPP protocol to validate users before allowing them access to server resources.

issue a valid certificate to a malicious third party. Therefore, security of password authentication on the web today practically relies on three assumptions: the security of the TLS channel, the ability of users to verify the validity of the channel and proper configuration and maintenance of the server.

Using PAKE. Password-Authenticated Key Exchange (PAKE) offers a relatively mature mechanism for password authentication that could be used on its own [4, 84, 57, 41] or as a building block in more complex protocols [89, 56, 35]. This cryptographic primitive provides a way to "bootstrap" a low entropy password into a strong cryptographic key without using public key infrastructure. Well-designed PAKE explicitly prevents *offline dictionary attacks* and can be used as a tool to diminish phishing attacks as well.

On its own, PAKE can be very useful in several different scenarios: A good example would be the file-transfer service, Magic Wormhole [114]. In this protocol a password is firstly generated and exchanged on the spot using an out-of-band channel. In the next step, this password is used in PAKE to establish a secure channel between two computers. Similarly, in the IoT scenario, EC-J-PAKE [54] has been recently proposed as a mechanism of choice for authorization of devices in the Thread protocol [52]. The third case where PAKE could be used is to replace self-signed certificates, which represent a popular but very often insecure way to avoid CAs[5]. Another potential use of PAKEs could be attributed to point-to-point protocols where third parties are not available.

As already mentioned, PAKE can be also used as a building block in higher-level protocols. For instance, Manulis et al. [89] proposed an alternative to the HTML-forms-over-TLS framework. In this newly proposed approach a secure channel provided by the TLS protocol run is composed and bound together with a PAKE execution that follows. The PKI-based server authentication, together with the mutual password-based client and server authentication, would prevent a wide range of attacks and enhance the security of the authentication process. It is yet to be seen if such a hybrid approach will be implemented in TLS in the future.

As a small note to a surprised reader who is wondering why PAKEs are still not quite included into the real-world applications, we point that for a long time PAKEs were encumbered with patent issues around EKE and SPEKE patents. This is probably one of the main reasons why PAKEs do not enjoy a wider use in systems

[5]Man-in-the-Middle (MitM) attacks are possible if a self-signed certificate is for the first time "introduced" to an honest client over an insecure environment (e.g. internet).

today. Additionally, as pointed in [40], integration of PAKEs into existing architecture and user re-education may occur as issues preventing the deployment of PAKEs. Finally, one should keep in mind that the use of PAKE would not solve all the existing problems with password-based authentication, such as weak password selection, issues around reuse of password across different websites, and problems related to password reset and recovery procedures.

Other Approaches. There exists a plethora of other mechanisms that try to implement secure authentication on the Internet today and they may or may not use passwords to do so. These range from different password managers, Single Sign-On services using Kerberos [94], decentralized authentication protocols such as OpenID [99], all the way to mechanisms for authorization such as OAuth [55].

1.3 Our Contributions

As already stated, the main purpose of PAKE is to allow secure authenticated communication over insecure networks between two or more parties who only share a low-entropy password. Although many different protocols that accomplish this aim have been proposed over the last two decades, only a few newcomers managed to find their way to real world applications - albeit lacking an intense and prolonged public scrutiny. As a step in the direction of providing one, this dissertation considers the security and efficiency of two relatively recently proposed PAKE protocols - Dragonfly and J-PAKE. In particular, we prove the security of a very close variant of Dragonfly employing the standard Find-then-Guess (FtG) model security of Bellare et al. [9] , which incorporates forward secrecy. Thus, our work confirms that Dragonfly's main flows are sound. The work presented in this thesis has contributed to the standardization process of the Dragonfly protocol in Internet Engineering Task Force (IETF) [57]. Furthermore, we contribute to the discussion by proposing and examining two variants of J-PAKE - which we call RO-J-PAKE and CRS-J-PAKE - that each makes the use of two less zero-knowledge proofs than the original protocol, at the cost of an additional security assumption. Our work reveals that CRS-J-PAKE has an edge in terms of efficiency over J-PAKE for both standard group choices: subgroups of finite fields and elliptic curves. The same is true for RO-J-PAKE, but only when instantiated with elliptic curves.

Taking a wider perspective, it is common practice that the secret key derived from a PAKE execution is used to authenticate and encrypt some data payload using

symmetric key protocols. Unfortunately, most PAKEs of practical interest, including three protocols considered in this thesis, are studied using so-called *game-based* models, which – unlike simulation models – do not guarantee secure composition *per se*. However, Brzuska et al. [26] have shown that a middle ground is possible in the case of authenticated key exchange that relies on PKI: the game-based models do provide secure composition guarantees when the class of higher-level applications is restricted to symmetric-key protocols. The question that we pose in this thesis is whether or not a similar result can be exhibited for PAKE. Our work answers this question positively. More specifically, we show that PAKE protocols secure according to the game-based Real-or-Random (RoR) definition of Abdalla et al. [3] allow for automatic, secure composition with arbitrary, higher-level symmetric key protocols. Since there is evidence that most PAKEs secure in the FtG model are in fact secure according to the RoR definition, we can conclude that nearly all provably secure PAKEs enjoy a certain degree of composition, one that at least covers the case of implementing secure channels.

1.4 Outline

We will begin in Chapter 2 with a review of some of the mathematical background necessary for the study of efficiency implications that different instantiations of our protocols can yield. Then the focus of the chapter will shift to the concept of provable security. Its essence will be discussed. Furthermore, we will present the formal description of the cryptographic constructions subsequently used in our protocols. Finally, we will finish this chapter with the survey of computational assumptions used to prove protocols' security.

Chapter 3 introduces password authenticated key exchange and game-based models which try to capture its desired security properties. More specifically, we will first formally define PAKE protocols and mention a few practical PAKEs of interest. Then, we will recall of the two most widely used game-based security models for PAKEs, Find-then-Guess (FtG) and Real-or-Random (RoR). As part of our contribution, the RoR model will be upgraded to accommodate for the analysis of composability of game-based secure PAKEs. We will conclude this chapter with a discussion on the relation between two models.

In Chapter 4, we consider the composition properties of game-based PAKE models. We will first introduce the limitations such models face in terms of composability and explain why the composition result from [26], shown in the context of AKE, does

not carry over to the PAKE setting. To improve this unfortunate state of affairs, we state the model requirements for both constituents, PAKE and symmetric key protocol, and then show how they naturally combine. The last section of this chapter demonstrates the suitability of the Real-Or-Random secure password key exchange protocol for composition with symmetric key protocols.

Chapter 5 contains two newly proposed PAKE protocols – the RO-J-PAKE and CRS-J-PAKE. These two protocols can be seen as more efficient variants of the J-PAKE protocol from [54]. Therefore, we will start this chapter by introducing the original J-PAKE design and protocol specifications. Furthermore, we describe our new protocols. The security proof - a crucial ingredient when proposing new cryptographic schemes - is provided for both protocols in the game-based RoR model. At the end, we also include some efficiency and deployment insights.

In 2013 the Dragonfly protocol was drafted in [61]. In Chapter 6, we analyzed a very close variant of Dragonfly protocol in FtG model. The proof of security that is provided in this chapter contributed to the protocol standardization in IETF in form of RFC7664 [57].

Chapter 7 offers concluding remarks where we discuss the obtained security results, highlight certain limitations of our approach and provide possible future research directions.

Chapter 2

Preliminaries

2.1 Introduction

One of the main goals of the modern cryptography is to enable secure communication between eager parties by using cryptographic schemes/protocols. A standard way to design a cryptographic scheme is to choose existing secure atomic primitives[1] (or rarely build a new one), and then, by using them as building blocks that offer fundamental properties, design a higher-level scheme which guarantee one or more high-level security properties. This composition should be made in such a way that the scheme can "inherit" security from these basic primitives. Typically, under the atomic primitive we assume either a *problem which is considered to be computationally hard* (e.g. the discrete log problem, the integer factorization problem) or a *secure cryptographic construction* that is believed to exist such as one-way function, pseudorandom generator, block cipher, compression function, etc. The problem that can arise with a cryptographic scheme design is that even if a good underlying atomic primitive is used, a poor design can result in an insecure scheme. The usual way to investigate whether a scheme inherits desired security properties from the underlying primitives is by means of provable security.

2.1.1 Previous Work

Two works that arguably mark the birth of modern cryptography are Diffie and Hellman's [38] and Rivest, Shamir and Adleman's [101]. Later, the concept of provable security was introduced by Goldwasser and Micali [48] in the context of probabilistic

[1] In this context the term "atomic" means that the primitive in question cannot be used alone to solve a specific cryptographic problem. Commonly, it is used as a building block of higher-level primitives.

encryption. In this groundbreaking paper, the authors propose the notion of indistinguishability which will be heavily used in our work. Following these early papers, a significant body of work has emerged over the last thirty years. For the nice introduction to provable security and core concepts that underlie the study of cryptography, the reader is referred to [70].

2.1.2 Organization

Section 2.2 will contain a review of some of the mathematical background necessary for the study of efficiency implications that different instantiations of our protocols can yield. Then, in Section 2.3 we will introduce the concept of provable security. Complexity-theoretic techniques will be discussed in 2.4. In Section 2.5, we will present auxiliary model assumptions that allow for an easier security analysis. Section 2.6 will contain the formal description of the cryptographic constructions that we will use in our protocols. Finally, in Section 2.7, we will present the survey of hardness assumptions that are used throughout our work in security reductions.

2.2 Mathematical Background

In this section, we first define notation and then provide a brief introduction to some of the concepts from group theory that will be used throughout this thesis. All of the material presented in this section can be found in similar form in existing literature, such as [70].

2.2.1 Notation

Let $\mathbb{N}$ denote the set of all natural numbers and $\mathbb{Z}$ denote the set of integers. For $n \in \mathbb{N}$, we denote the set of n-bit strings by $\{0,1\}^n$. The set of all finite, arbitrary-length binary strings is denoted by $\{0,1\}^*$. The concatenation of two bit strings x and y is denoted by $x|y$. We denote the output y of a *deterministic* function f on input $x_1, \ldots, x_n$ as $y := f(x_1, \ldots, x_n)$. Similarly, $y := x$ denote deterministic assignment of the value x to the variable y. We write $y \leftarrow F(x_1, \ldots, x_n; r)$ to denote probabilistic process in which random coin r is chosen uniformly at random and to y is assigned the output of $F(x_1, \ldots, x_n; r)$. Similarly, let $a \leftarrow A$ denote selecting a uniformly at random from the set A. Also, for both deterministic and probabilistic function f, the predicate $y = f(x_1, \ldots, x_n)$ will be true if the output of function f on input $x_1, \ldots, x_n$ is equal to the value y.

2.2.2 Groups

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

Closure: *For any two elements g and h from $\mathbb{G}$, $g \diamond h$ is element in $\mathbb{G}$.*

Presence of a neutral element: *There exists a neutral or an identity element $e \in \mathbb{G}$ such that for all $g \in \mathbb{G}$ holds that $e \diamond g = g = g \diamond e$.*

Presence of inverses: *For all $g \in \mathbb{G}$ there exists an inverse element $h \in \mathbb{G}$ such that $g \diamond h = e = h \diamond g$.*

Associativity: *For all $g, h, j \in \mathbb{G}$ holds that $(g \diamond h) \diamond j = g \diamond (h \diamond j)$.*

Throughout this thesis, we will assume that group operation is known and we will represent the group ($\mathbb{G}$, $\diamond$) only with the set $\mathbb{G}$. We say that a group $\mathbb{G}$ is finite if it has a finite number of elements. The number of elements in $\mathbb{G}$, or shortly the *order* of the group, is denoted by $|\mathbb{G}|$. Given a group $\mathbb{G}$ under a binary operation $\diamond$, a set $\mathbb{H} \subseteq \mathbb{G}$ is a subgroup of $\mathbb{G}$ if $\mathbb{H}$ itself forms a group under the operation $\diamond$. Since in this thesis we will exclusively work with finite, *abelian* groups, we introduce them below.

Definition 2.2. *A group $\mathbb{G}$ with operation $\diamond$ is* ***abelian*** *(also called commutative) if the the following axiom holds:*

Commutativity: *For any two elements g and h from $\mathbb{G}$, $g \diamond h = h \diamond g$.*

Note that in case of multiplicative operation, instead of $\diamond$ we will be using $\cdot$ notation. Furthermore, when using multiplicative notation, we will denote group exponentiation by g^n, instead of $g \cdots g$. In the expression $h = g^n$, we call n the discrete logarithm of h to base g. Then,

Theorem 2.3. *Let $\mathbb{G}$ be a finite group of order $w = |\mathbb{G}|$. Then for every element g from the group $\mathbb{G}$ it holds that $g^w = 1$.*

Cyclic Groups. Now we will introduce a notion of cyclic groups since the majority of hardness assumptions used in cryptography rely on the cyclic property of the group.

Definition 2.4. *Let $\mathbb{G}$ be a finite group of order w. The group $\mathbb{G}$ is* ***cyclic****, if there exists a generator $g \in \mathbb{G}$ such that $\{g^0, g^1, \ldots, g^{w-1}\} = \mathbb{G}$.*

Loosely speaking, if $\mathbb{G}$ is a multiplicative cyclic group and g is a generator of the group G, then from the definition of cyclic groups, we know every element h in $\mathbb{G}$ can be written as g^n for some n. One can now define the problem that underlies major part of modern cryptography: given a group $\mathbb{G}$, a generator of the group g and an element h of $\mathbb{G}$, one needs to find the discrete logarithm of h to the base g in the group $\mathbb{G}$. Note however that the discrete logarithm problem is not "hard" in all cyclic groups.

Prime Order Groups. In this thesis, we will use groups of prime order, which are preferred in practice for several reasons. First of all, any group $\mathbb{G}$ of prime order is cyclic. Second, as a consequence of Pohlig-Hellman algorithm, the discrete log problems are considered to be hardest in such groups. Third, from the efficiency perspective, it is easy to find a generator of the group since all elements of group $\mathbb{G}$ (except the identity) are generators of $\mathbb{G}$. Fourth, the only subgroups of $\mathbb{G}$ are trivial subgroups $\{0\}$ and $\mathbb{G}$. Lastly, in proofs sometimes it is easier to reduce the security of some scheme or construction to *decisional Diffie-Hellman* problem (see Section 2.7) when the group order is prime.

Subgroups of $\mathbb{Z}_p^*$. Groups of form $\mathbb{Z}_p^*$[2], where p is a prime and the group operation is multiplication modulo p, give rise to one class of cyclic groups in which discrete logarithm problems are considered to be hard. This is achieved by working with q-order subgroup of $\mathbb{Z}_p^*$, where $p = rq + 1$ and p and q are both primes and r is called co-factor.

Theorem 2.5. *Let $p = rq + 1$, where p and q are primes. Then*

$$\mathbb{G} = \left\{[h^r \ mod \ p] \mid h \in \mathbb{Z}_p^*\right\}$$

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

This result can be generalized. Namely, the discrete problem is believed to be hard in a multiplicative group of the finite field of a large characteristic when the polynomial representation is used. We will use such algebraic structure in Chapters 5 and 6.

[2]An upper-script $(^*)$ shows that the zero element is excluded from a set.

Elliptic Curves (EC). In addition to the groups that are directly based on modular arithmetic, such as $\mathbb{Z}_p^*$ or the multiplicative group of s finite field, another class of groups that are useful in cryptography are groups that consist of points on elliptic curves. These groups offer more efficient implementation than previous candidates while providing the same level of security. As our work is only slightly concerned with elliptic curve groups, we keep the discussion minimal; a more extensive discussion can be found in Hankerson et al. [53].

Definition 2.6. *Let p be a prime such that $p \geq 5$. An **elliptic curve** E is of the form*

$$y^2 = x^3 + Ax + B \ mod \ p,$$

where $A, B \in \mathbb{Z}_p$ are constants and equation $4A^3 + 27B^2 \neq 0 \ mod \ p$ holds.

Let $E(\mathbb{Z}_p)$ include the set of pairs $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ that fulfill equations from the definition above together with the point of infinity $\mathcal{O}$. A pair that is the element of $E(\mathbb{Z}_p)$ represents a point on the elliptic curve E. Most importantly, the set of points $E(\mathbb{Z}_p)$ with specific additive operation form an abelian group, where $\mathcal{O}$ is the identity element. Further, if few conditions are fulfilled, the decisional Diffie-Hellman problem is believed to be hard in elliptic curve groups of large prime order.

2.3 Provable Security Paradigm

Fundamentally, the goal of provable security is to provide a strong mathematical guarantee that a cryptographic scheme cannot be broken by a class of attackers in a specified security model that tries to capture the real-world scenarios. In practical terms, the provable security approach allows us to prove the security of higher level scheme – such as key exchange protocol – under some well-established assumptions which include the hardness of the discrete logarithm type problems coupled with the security of the underlying digital signature scheme and/or hash function, for instance. This means that as a result of applying provable security framework we actually (in most cases) *do not* provide an absolute proof of security. What is provided, in fact, is a reduction of the security of the scheme to the security of some underlying atomic primitive. Therefore, the term that maybe better reflects the essence of this approach is *reductionist approach*.

The steps one should take when using such reductionist approach are the following:

1. Formally define a security notion one wants to achieve for a particular scheme or primitive of interest.

2. Choose a formal adversarial model that is suitable for a concrete security goal.

3. Clearly state and define any assumption or atomic primitive upon which the security of scheme relies.

4. Once all three previous steps are fulfilled, one has to exhibit a security reduction which shows that the only practical way to defeat the scheme is to break the underlying assumption (or atomic primitive).

Security proofs derived in such a way give the guarantee that an assumption is enough for security: if some adversary can break the higher-level scheme, one can use such adversary to break the underlying assumption or atomic primitive. This practically means – under the assumption that the security reduction (or proof) is correct – that if there exists some security flaw in the scheme, there must be a weakness in the underlying atomic primitive as well. And vice-versa, if we believe that the atomic primitive is secure, then we will know that the scheme must be secure with respect to the desired security notion. An additional benefit of the reductionist approach is that it makes cryptanalysis of the schemes simpler, since its focus after having a proof of security should be placed on the atomic primitive.

To summarize, there are two principal aims of the provable security approach. The first is associated with the introduction of notions and their definitions which practically entails classification of protocols and atomic primitives, while the second is related to providing the security arguments. Most of the security arguments in this thesis will be based on security reductions.

2.4 Complexity Theory Techniques

When analyzing some cryptographic protocol or a primitive, one needs to quantify the level of security such scheme may provide. In the ideal case, a cryptographic scheme would be *information-theoretically secure* i.e., it could not be broken even when the adversary has unlimited computing power. However, in most of the real-world applications, the trade-off between security and efficiency would prevent such schemes to become feasible. Thus, to model practical scenarios today, cryptographers are using *computational* security approach, which takes into account the computational power relevant adversaries may have, and admit a negligible probability of failure. In other

words, when using computational approach – in contrast to information-theoretical approach – the security is only guaranteed in presence of *efficient* adversaries that run for some feasible amount of time, and that have some very small probability of success. Depending on how detailed we define what *efficient* adversary means, we distinguish between two types of computational approach: the asymptotic security and the concrete security.

2.4.1 Asymptotic Security

Borrowed from the complexity theory, *asymptotic security* approach is widely used when studying the security of cryptographic schemes. In this framework, an algorithm is considered to be efficient when it runs in polynomial time[3]. Furthermore, a *security parameter* – in this thesis denoted by the Greek letter κ and sometimes provided in unary notation (1^κ), (often implicitly) parametrizes all of the (probabilistic) algorithms that are involved in the analysis: honest parties, the adversary, a cryptographic scheme under analysis, a key generation algorithm and even the security reduction. The inclusion of the security parameter allows us to talk about efficient algorithms, e.g. the adversary's running time and success probability can be represented as a function of security parameter. Therefore, the *efficient* adversary matches with probabilistic algorithm running in time polynomial in the security parameter. Similarly, a practical scheme needs to be designed with polynomial-time algorithms.

To quantify very small success probability that is allowed for the adversary in computational security framework, we use the term *negligible success probability* which stands for a probability that is smaller that any inverse polynomial in the security parameter. Finally, a security result provided by the asymptotic approach would typically take the following form: A cryptographic scheme is secure with respect to some security notion if any probabilistic polynomial-time adversary succeeds in breaking that security notion with at most negligible probability.

2.4.2 Concrete Security

While the polynomial-time approach is quite favorable in the theoretical domain, in practice it is more desired (or even crucial) to provide concrete numbers, which bound the maximum success probability of any adversary when investing some specific amount of computational power. This framework is called *concrete security* framework [8] and it captures the quantitative nature of security. Let us illustrate

[3]Note that an efficient algorithm is able to call other efficient algorithms as subroutines.

the difference between concrete and asymptotic security on the following example. Consider the key generation algorithm $KGen$. When using asymptotic approach key generation algorithm would take as input security parameter $KGen(1^\kappa)$ whereas the key-length provided as a fixed value would be the parameter of choice when using concrete security approach. As another point of comparison, a security result yielded by the concrete security approach would typically take the following form: A cryptographic scheme is (t, ε)-secure with respect to some security notion if any adversary running in time t succeeds in breaking that security notion with probability at most ε. The reason asymptotic security approach is more widely used in the academic community is because of extra difficulty to obtain precise concrete guarantees.

2.4.3 Security Resistance and Attacks

In the provable security framework, attacks and security resistance are the complements of each other. Attacks measure the degree of insecurity while quantitative bounds measure the degree of security. More precisely, while the proof of security provides a lower bound, cryptographic attacks provide an upper security bound on the complexity of breaking scheme under some assumption. When these two bounds meet, the security property of the scheme is identified and the bound is declared as *tight*.

2.4.4 Reductions

As we mentioned before, cryptographers usually use the reductionist approach to prove that their protocols are secure in the computational model. The security reduction from problem P_1 to problem P_2 can be seen as an algorithm that solves problem P_1 when running as subroutine an algorithm solving problem P_2. In our work, we will consider efficient security reductions, i.e. polynomial-time reductions.

2.5 Model Assumptions

The difficulty of exhibiting a security proof of higher-level protocols or primitives solely under cryptographic hardness assumptions has forced cryptographers to introduce certain assumptions on the environment in which a scheme executes. These include idealized assumptions, setup assumptions, and possibly restricted models of computation. Aforementioned assumptions are introduced to provide an additional

power to the security reduction over the environment when reasoning about the security of cryptographic protocols.

Probably the most influential idealized assumption used in cryptography is *Random Oracle* (RO), a model in which an object of idealization is a hash function [11]. Other cryptographic objects that were assumed to be ideal in literature range from block ciphers [9, 5], compression functions [5], all the way to to permutations [112], etc. When mentioning setup assumptions, one usually means on *Common Reference String* (CRS). Roughly, in CRS model it is assumed that all the participants have access to a string that is drawn according to some specified distribution. Lastly, under the term restricted model of computation, we assume a model in which one considers only generic algorithms, those that can not exploit the properties of the representation of the algebraic structure under consideration. As examples of such models, we include those of Shoup [108] and Paillier and Vergnaud [95], the latter being more relevant to our work.

Terminology. The model of computation in which the adversary is only limited by the amount of time and computational power available – is the most desirable one. As we pointed before, cryptographic schemes are often based on cryptographic hardness assumptions[4]. Those schemes whose security reduction is possible using only cryptographic hardness assumptions are said to be secure in the *plain model*. Although a proof in the plain model brings stronger security guarantees than other techniques, usually in practice it is quite difficult to complete this type of proof. This is especially true when the computational efficiency of the resulting cryptographic scheme is paramount. One speaks about a *standard model* if the CRS assumption is admitted in addition to hardness assumptions. In case RO assumption is used, one is dealing with the *random oracle model*, or more general the *ideal model*. Note that it is possible that one model encompasses all three types of the assumptions on the environment.

2.5.1 Random Oracle Model (ROM)

In order to allow the design of more practice-oriented provably-secure cryptographic schemes, Bellare and Rogaway formally introduced the random oracle framework in [11]. In this framework, hash functions (see below) are modeled as public, random functions - with co-domain $\{0,1\}^n$ or some particular group - that the adversary has

[4]Under cryptographic hardness assumptions we consider an assumption on the hardness of the underlying problem (e.g. the discrete log problem, the integer factorization problem).

query access to. Answers to new queries are selected randomly, while answers to previously asked queries are repeated. Notice that by using prefixes one can generate multiple random oracles from a single random oracle. More formally,

Definition 2.7. *A **random oracle** is a public hash function that maps inputs of arbitrary size to outputs of finite size, or* $R : \{0,1\}^* \rightarrow \{0,1\}^n$*, where the outputs are drawn uniformly at random from the range space and accessible by all algorithms in a black-box manner.*

When using the random oracle model one needs to be aware that the random oracle assumption is the strongest assumption possible for hash functions. As a consequence, the security guarantees provided by the random oracle model are not as strong as those obtained in the plain or standard model. For more details on how random oracle is used in the proofs of security check Chapter 5 and see Figure 5.4.

What are the advantages of this approach? The main advantage of the random oracle model is that it allows us to prove efficient-in-practice cryptographic schemes secure. Furthermore, even though the random oracle assumption is strong, the results obtained in the random oracle model provide valuable security guarantees (e.g. exclusion of certain generic attacks, the absence of security flaws in the design, etc.). For a nice discussion on justification, history, and power of random oracle assumption, we refer the reader to [75].

2.5.2 Common Reference String (CRS)

As with the RO model, a CRS model is very useful when it comes to designing cryptographic schemes and trying to prove them secure. The CRS model has been introduced by Blum et al. [15] to allow for the construction of Non-Interactive Zero-Knowledge (NIZK), which was shown to be impossible in the plain model [15, 16]. Another instance where CRS turned out to be valuable is when constructing Universally Composable (UC) primitives and protocols [27]: constructing UC bit commitments [28] and UC secure PAKEs [29] is known to be impossible in the plain model. Let us now define CRS model.

In the CRS model, a public, trusted value – called the common reference string – is selected at setup time according to some predefined distribution and given to all participants and the adversary, under the assumption that no extra information about the common reference string is available to anyone. More formally,

Definition 2.8. *Let T be an efficient probabilistic algorithm that outputs string σ of finite size according to arbitrary probability distribution $\mathcal{C}$ when given as input a security parameter κ. Then, σ is called a* ***common reference string*** *if it is made available to all algorithms.*

This will simply mean that all algorithms involved will have CRS as an extra input. In practice, one can think of it as a common parameter that is defined in the setup phase, such as the description of a group. However, the true power of CRS assumption comes to life when making security arguments: Namely, the security reduction algorithm (or simulator) may associate properly looking CRS with an underlying *secret trapdoor* by using unnoticeably different CRS generation algorithm. This gives the simulator that is in possession of the secret trapdoor extra abilities during the security reduction. For a nice example of CRS assumption use, which is relevant to our work, we point the reader to [71]. We will also use this assumption to prove CRS-J-PAKE secure in Section 5.5.3.

2.5.3 Restricted Models of Computation

As Maurer points out in [90], some restricted models of computation can be meaningful in cryptography. One that is usually used is the generic model, which assumes that the representation of the elements of the algebraic structure under consideration (usually group) can not be exploited. By the term representation we mean the way group elements are encoded in order to allow computation over them. Remember also that the cyclic group of p elements is typically instantiated with an additive group of integers, a multiplicative group of integers or a subgroup of an elliptic curve group. Even though these groups are isomorphic, they have very different computational properties. In cryptography, we usually use bitstrings to encode the elements of a group. A generic algorithm is an algorithm that works independently of the group's representation and only admits two operations that any representation inherently has: equality testing and a total order relation.

Note that by using restricted models we can abstract away from the specific group and work over some generic, unspecified group. This approach works well with discrete logarithm problems and is not applicable to those that are RSA based.

Generic Group Model (GGM). The first to apply this model in cryptography was Shoup [108]. Interestingly, in his model he assumes that access to group elements

is via a randomly selected representation. This differs from Maurer's more general approach in [90], which allows for certain additional information from the representation of the elements (e.g. the availability of a Decisional Diffie-Hellman (DDH) oracle)[5].

In generic group model, it is assumed that in the case of a group in which the discrete logarithm problem is intractable, an adversary can perform group operations only by means of an oracle that he has access to. The oracle is constrained to return correct results. On his own, the adversary can only test for equality of two elements.

To see how strong GGM model assumption really is, let us compare an oracle for an elliptic curve group and an oracle for a generic group. Since elliptic curve point and its inverse share the same x-coordinate, two oracles will be easily distinguished from each other. However, when looking at security consequences we see that the one who tries to exploit the representation of group elements does not perform better than *generic* algorithms (i.e. Pohlig-Hellman and Pollard's algorithm[6]) - in case of the discrete logarithm problem on general[7] elliptic curves [23]. An example of *non-generic* algorithm for discrete logarithm problem is *index calculus algorithm* that applies on subgroups of $\mathbb{Z}_p^*$, where p is prime.

As a final note, a security result for some scheme in GGM implies that one can break the scheme only by exploiting particular representation of the group elements and not by using generic algorithms.

Algebraic Model. In our work, we will use the algebraic model from Paillier et al. [95]. In algebraic model, similarly to GGM, an adversary can perform group operations on the elements of the group $\mathbb{G}$ only via oracle(s). These group operations include:

1. Equality testing: $ET(g, h) \to 1$ if $g = h$ and 0 otherwise.
2. Group operation: $GO(g, h) \to g \cdot h$.
3. Exponentiation operation: $EO(g, \alpha) \to g^{\alpha}$.

[5]Introducing this oracle can help answering the following question: Does a DDH-oracle help in computing discrete logarithms?

[6]Although Pohlig-Hellman algorithm and Pollard's algorithm are generic, in order to use their full power, different encoding of group elements should be used: Pohlig-Hellman takes the advantage of bitstring encoding of group elements which facilitates the use of fast sorting and searching techniques, while on the other hand, efficiency of Pollard's algorithm relies on the assumption that group elements use a randomly selected encoding.

[7]Note that for some particular elliptic curve groups different group representation can be exploited.

4. Inverse operation: $IO(g) \rightarrow g^{-1}$.

This means, for instance, that the adversary can not set the last bit of a group element to the value of his choice (when using bitstring representation), which in practice could help him to gain some additional information. To make it clear, the adversary can apply arbitrary operations on all the other data types, except the group elements in $\mathbb{G}$. In contrast to the generic group model of Shoup [108], an algebraic model of Paillier et al. does not make any assumption on the representation of group elements. Also, one does not need to assume that the discrete logarithm is hard. Therefore, the algebraic model can be considered as weaker than the generic group model. Again, a security result for some cryptographic scheme in the algebraic model tells us that if the adversary wants to break the scheme, he must find some way to exploit intricate, non-algebraic relations within the group $\mathbb{G}$.

At the end of this section, it should be noted that schemes that have proofs of security in idealized or restricted models are not necessarily weak, but the proof of security should be considered only as a heuristic guide to the security of the algorithm.

2.6 Cryptographic Building Blocks

In this section, we introduce those cryptographic constructions that will be used as building blocks in higher-level protocols our thesis is concerned about, namely Password Authenticated Key Exchange (PAKE). We will start this section by introducing hash functions and the concept of computational randomness extractor. Then we will deal with zero-knowledge proofs, or more specifically with certain types of non-interactive zero-knowledge proofs. Lastly, we will present Schnorr signatures.

2.6.1 Hash Functions

In order to eliminate an ambiguity that exists between cryptographic hash functions (in which we are interested in) and general-purpose hash functions, we start with the formal definition of the former[8].

Definition 2.9. *A **hash function** is a deterministic function that maps an input of finite arbitrary size to an output of finite fixed size n. Formally, H: $\{0,1\}^* \rightarrow \{0,1\}^n$.*

[8]Note that in our work we will be dealing with keyless hash function. In contrast, keyed hash functions take as input an additional key k and are denoted as $H_k(M)$ or $H(k, M)$. Note that in practice the key is usually chosen in advance and omitted.

In his PhD thesis [91], Merkle defined the three main security notions of hash functions. These basic notions of security were revisited and formalized in a wider context in [102]. At this point, commonly used informal definitions of the three main security notions are provided:

- **collision resistance** (Coll) – it is hard to find any two distinct inputs M and M' which hash to the same output, such that $H(M) = H(M')$.
- **second-preimage resistance** (Sec) – it is hard to find any second input which has the same output as any specified input, i.e., given M, to find a second-preimage $M' \neq M$ such that $H(M) = H(M')$.
- **preimage resistance** (Pre) – it is hard to find any input which hashes to that output, i.e., to find any preimage M' such that $H(M') = Y$.

Since we will be exclusively concerned with the notion of collision resistance in this thesis, we will only present the expected security of hash function relative to this notion. For more details on other two notions we refer reader to [112].

Collision Resistance. Here we want to show the security results that hold for hash functions in general, which means that we do not want to focus on any particular hash design. In order to achieve this, we will consider a hash function which act as the random oracle (see 2.5.1). Interestingly, we can equate the problem of finding a collision on a hash function with the *birthday paradox*. If we map the birthday paradox to our collision problem, such that our range length is 2^n possible values, it is clear that after $\sqrt{2^n} = 2^{n/2}$ queries to hash function, collision is going to be found with the probability higher than 50%. The same problem can be seen from a different angle: If an adversary is trying to find a collision, he can start by sending queries and receiving outputs from the random oracle. If the outputs of distinct random oracle queries coincided, he would find a collision. After the adversary sends q queries to random oracle, he can make a list which consists of $q(q-1)/2$ random oracle output pairs. Due to the bit-length of the random oracle output, each pair results in collision with probability 2^{-n}. This means that the probability of finding collisions between random oracle outputs grows almost quadratically with the number of queries adversary asks and is inversely proportional to the bit-size of random oracle output. Therefore, the adversary's probability of success is upper bounded with roughly $\frac{q^2}{2^n}$.

2.6.2 Computational Randomness Extractor

Apart from collision resistance, cryptographic hash functions exhibit some other properties that are very useful for different practical purposes. For instance, they are considered to provide randomness-like properties such as the independence of input from output, or output unpredictability in case of partially unknown input. Thus, one of the relevant real-world applications of secure cryptographic hash function, typically keyed one, is to act as a Key Derivation Function (KDF). An objective of the key derivation function is to derive one or more secret keys from a shared secret value which can originate from a variety of sources: it can come as a result of a key exchange protocol, or it can be a previously established secret such as a master key, a password, or a passphrase. KDF can be also used to transform secret in a required format or to expand keys such they reach a desired length.

In our case, we will be using KDF to convert a group element that is the result of a Diffie-Hellman style key exchange into a symmetric key that can be used with a block cipher. In security proofs, especially those that are relevant in the context of game-based password key exchange, a hash function that is used for key derivation is usually modeled as a random oracle. However, it was shown in [1] that in case of J-PAKE protocol [54] such heuristic assumption is too strong and that a computational randomness extractor for random group elements from [76] is sufficient. Let us define the requirements for such derivation function.

Definition 2.10. *A **randomness extractor** ext_R is a function that for some $t \geq 0$ maps $\{0,1\}^t \times \mathbb{G} \rightarrow \{0,1\}^k$. The extractor is said to be secure if a polynomial-time adversary $\mathcal{A}$'s advantage in distinguishing $ext_R(e,r)$ – where (e,r) is randomly sampled from $\{0,1\}^t \times \mathbb{G}$ – from a random bit-string in $\{0,1\}^k$ given e is negligible.*

For more details on a computational randomness extractors, see [76].

2.6.3 Zero-Knowledge Proofs

Informally, a Zero-Knowledge proof (ZK), as defined by Goldwasser, Micali and Rackoff [49], is an interactive probabilistic protocol between two parties, a prover and a verifier, that allows the prover to convince the verifier that a given statement is true, without leaking any other information except the fact that the statement is indeed true.

Non-Interactive Zero-Knowledge (NIZK). In our work, we are interested in NIZK proofs that are a variant of zero-knowledge proofs in which interaction between a prover and a verifier is deemed unnecessary. In the random oracle model, NIZK proofs can be obtained from interactive zero-knowledge proofs using the Fiat-Shamir heuristic [42]. In [15], Blum et al. showed that computational NIZK can be also achieved by using a common reference string.

Let us now recall definitions from [50] that are extended in [1] to the case of labeled Non-Interactive (NI) proof systems.

Definition 2.11. *Let $\mathcal{R}$ be an efficiently computable relation with a binary output and two inputs (x, w), where x and w are called the statement and the witness, respectively. Let* L *be the NP-language with respect to $\mathcal{R}$ that consist of statements* $\mathrm{L} := \{x \mid \exists w,\ \mathcal{R}(x, w) = 1\}$. *A common reference string σ is produced after running setup algorithm T. Then, on input (σ, x, w), where relation $\mathcal{R}(x, w) = 1$, the prover produces a proof $\pi \leftarrow PK(\sigma, x, w, l)$ for some label l. The verifier, or anyone holding σ, x, π, and l, can verify the proof by running algorithm $VK(\sigma, x, \pi, l)$, which outputs* 1 *if the proof is valid, and* 0 *otherwise. We call (T, PK, VK) a labeled* ***NI argument*** *or proof system for a relation $\mathcal{R}$, where k is security parameter and ε is negligible, if the following two conditions hold:*

Completeness: *For all adversaries $\mathcal{A}$ we have:*

$$Pr\big[\ \sigma \leftarrow T(1^\kappa);\ (x, w, l) \leftarrow \mathcal{A}(\sigma);\ \pi \leftarrow PK(\sigma, x, w, l) : VK(\sigma, x, \pi, l) = 1 \text{ if } \mathcal{R}(x, w) = 1\ \big] \approx 1. \tag{2.1}$$

Soundness: *For all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$ we have:*

$$Pr\big[\ \sigma \leftarrow T(1^\kappa);\ (x, \pi, l) \leftarrow \mathcal{A}(\sigma) : VK(\sigma, x, \pi, l) = 1 \text{ if } x \notin \mathrm{L}\ \big] \leq \varepsilon. \tag{2.2}$$

Loosely speaking, the completeness property captures the probability that an honest verifier accepts a correct proof. Proof systems that we use in Section 5 will satisfy perfect completeness i.e., a correct proof will be accepted with probability 1. On the other hand, the soundness property captures the probability that the verification procedure accepts an incorrect proof of the statement as true (under condition that the statement is not in language). For our purpose, a non-interactive argument (computational proof system) is sound if there exists no polynomial time adversary $\mathcal{A}$ that can prove a false statement with non-negligible probability. The perfect soundness would be achieved when the probability that verifier accepts the false proof is equal to 0. Next property that we recall is knowledge extraction.

Definition 2.12. *We call* (T, PK, VK) *a labeled* ***NI Proof of Knowledge*** *(PoK) for a relation* $\mathcal{R}$ *if there exist a knowledge extractor* $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ *with the following properties:*

Knowledge Extraction: *For all adversaries* $\mathcal{A}$ *we have:*

$$Pr\big[\sigma \leftarrow T(1^\kappa) : \mathcal{A}(\sigma) = 1\big] \approx Pr\big[(\sigma, \xi) \leftarrow \mathcal{E}_1 : \mathcal{A}(\sigma) = 1\big]. \tag{2.3}$$

Also, for all adversaries $\mathcal{A}$ *we have:*

$$\begin{aligned} Pr\big[\ (\sigma, \xi) \leftarrow \mathcal{E}_1(1^\kappa);\ (x, \pi, l) \leftarrow \mathcal{A}(\sigma);\ w \leftarrow \mathcal{E}_2(\sigma, \xi, x, \pi, l) : \\ VK(\sigma, x, \pi, l) = 0 \text{ or } \mathcal{R}(x, w) = 1\ \big] \approx 1. \end{aligned} \tag{2.4}$$

where $\mathcal{S}'(\sigma, \tau, x, w) = \mathcal{S}_2(\sigma, \tau, x)$ *for* $\mathcal{R}(x, w) = 1$.

This property intuitively captures the following: if the prover or the adversary can succeed in making the verifier accept, then there exist another algorithm called the extractor, that can communicate with the prover and can produce a valid witness from any accepting proof. In security proof, usually for extraction to work the extractor has to run the prover several times using the same randomness. This technique is called *rewinding*, and is not possible under normal circumstances since it could jeopardize the zero-knowledge property (see below). Now we can provide formal definition for Non-Interactive Zero-Knowledge (NIZK) proof (system).

Definition 2.13. *We call* (T, PK, VK) *a labeled* ***NIZK proof*** *for a relation* $\mathcal{R}$ *if there exist a simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *with the following property:*

Zero-Knowledge: *For all polynomial-time adversaries* $\mathcal{A}$ *we have:*

$$Pr\big[\sigma \leftarrow T(1^\kappa) : \mathcal{A}^{PK(\sigma,\cdot,\cdot,\cdot)}(\sigma) = 1\big] \approx Pr\big[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa) : \mathcal{A}^{\mathcal{S}'(\sigma,\tau,\cdot,\cdot,\cdot)}(\sigma) = 1\big] \tag{2.5}$$

where $\mathcal{S}'(\sigma, \tau, x, w) = \mathcal{S}_2(\sigma, \tau, x)$ *for* $\mathcal{R}(x, w) = 1$.

To clarify the definition, an NIZK proof is zero-knowledge if from the adversary's perspective simulated proofs (proofs formed by the simulator) are indistinguishable from real proofs (proofs formed by the honest prover). In other words, the proof system is zero-knowledge if any information a verifier could learn by interacting with the honest prover, the verifier could also learn itself. Finally, perfect zero-knowledge is achieved if for all (x, w) pairs such that $\mathcal{R}(x, w) = 1$ two probability distributions are identical.

Next property that we will recall is called *simulation soundness*, which ensures that the soundness of the proof system remains intact even in case the adversary sees the simulation of a proof for a false statement.

Definition 2.14. *We call* (T, PK, VK) *a* ***simulation sound*** *(labeled)* ***NIZK proof*** *for a relation* $\mathcal{R}$ *if there exist a simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *with the following property:*

Simulation Soundness: *For all polynomial-time adversaries* $\mathcal{A}$ *we have:*

$$Pr\big[\ (\sigma,\tau) \leftarrow \mathcal{S}_1(1^\kappa);\ (x,\pi,l) \leftarrow \mathcal{A}^{\mathcal{S}_2(\sigma,\tau,\cdot,\cdot)}(\sigma) : (x,\pi,l) \notin \mathcal{L}_{sim} \\ \text{and } x \notin \mathrm{L} \text{ and } VK((\sigma,x,\pi,l)) = 1\big] \leq \varepsilon, \tag{2.6}$$

where $\mathcal{L}_{sim}$ *is the list that collects query-response pairs* $((x_i, l_i), \pi_i)$ *of size* i*.*

To put differently, an NIZK proof is simulation sound if an adversary is not able to prove any false statement even after seeing simulated proofs of arbitrary statements.

Simulation-Sound Extractable NIZKPoK (SSE-NIZKPoK). The last property that we recall here combines two previously defined properties: zero-knowledge and knowledge extraction. We will call such system a Simulation-Sound Extractable NIZK Proofs of Knowledge (SSE-NIZKPoK).

Definition 2.15. *Consider* (T, PK, VK) *to be a labeled* ***SSE-NIZKPoK*** *for a relation* $\mathcal{R}$ *if there exist a simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *and an extractor* $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$*. Let* $\mathcal{SE}_1$ *be an algorithm that, on input* 1^κ*, outputs* (σ, τ, ξ) *such that* (σ, τ) *is distributed identically to the output of* $\mathcal{S}_1$*. An NIZK proof of knowledge* (T, PK, VK) *is* ***simulation sound extractable*** *if the following property holds:*

Simulation Sound Extractability: *For all polynomial-time adversaries* $\mathcal{A}$ *we have:*

$$Pr\big[\ (\sigma,\tau,\xi) \leftarrow \mathcal{SE}_1(1^\kappa);\ (x,\pi,l) \leftarrow \mathcal{A}^{\mathcal{S}_2(\sigma,\tau,\cdot,\cdot)}(\sigma,\xi);\ w \leftarrow \mathcal{E}_2(\sigma,\xi,x,\pi,l) : \\ (x,\pi,l) \notin \mathcal{L}_{sim} \text{ and } \mathcal{R}(x,w) = 0 \text{ and } VK((\sigma,x,\pi,l)) = 1\big] \leq \varepsilon, \tag{2.7}$$

where $\mathcal{L}_{sim}$ *is the list that collects query-response pairs* $((x_i, l_i), \pi_i)$ *of size* i*.*

Loosely speaking, ZK property ensures that simulated proofs are indistinguishable from real one, while knowledge extraction guarantees that there exists an extractor algorithm that can extract a witness from any adversary-generated proof (even if the adversary can see simulated proofs). Therefore, to defeat the property, an adversary that is allowed to see simulated proofs (of potentially fake statements) must provide valid looking proof for adaptively chosen statements in such a way that an

extractor cannot obtain witnesses. Groth indicated in [50] that the simulation sound extractability implies simulation soundness. Namely, if the extractor can produce a valid witness from the adversary's proof, then the statement must belong to the language in question.

2.6.4 Schnorr Signatures (Proofs of Knowledge)

In 1989, Schnorr [105] presented his very efficient identification protocol which provides means to prove knowledge of the discrete logarithm of a publicly known group element without revealing it (exponent). In the same paper, Schnorr showed that by applying Fiat-Shamir transform [42] (with respect to a hash function) to his identification scheme one can obtain non-interactive signature scheme - which is known as Schnorr signature. Using the forking lemma and modeling a hash function as the random oracle allowed Pointcheval and Stern [97] to prove the security of Schnorr signature against chosen message attacks. Let us now formally define Schnorr signature scheme.

Definition 2.16. *A **Schnorr signature** scheme consists of three polynomial-time algorithms:*

Key generation algorithm: *Select a random $x \leftarrow \mathbb{Z}_q$ and compute $Y := g^x$. The secret key is x, while $Y \in \mathbb{G}$ acts as the public key.*

Signing algorithm: *For a given message $m \in \{0,1\}^*$ and label l, the signature algorithm Sig picks a random $k \leftarrow \mathbb{Z}_q$, computes $R := g^k$, $c := H(m, R, l)$ and $s := k + cx \bmod q$. On input $Sig(m, l)$ algorithm outputs (s, c).*

Verification algorithm: *If $H(m, g^s Y^{-c}, l) = c$, the verification algorithm Ver on input $(m, (s, c), l)$ returns 1. In case equality does not hold, Ver returns 0.*

Application. In Section 5, we will use Schnorr proofs of knowledge as the instantiation of SSE-NIZKPoK in our protocols. As can be seen from Definition 2.15, simulation sound extractability is typically enabled in NIZK proof systems by generating two trapdoors for some additional CRS at the setup time. However, in Schnorr signature scheme such trapdoors are not directly available. Fortunately, it was shown in [1] that Schnorr signature scheme satisfies both properties under the following conditions: ZK stems from the programmability of the random oracle, while for knowledge

extractability the adversary has to be assumed *algebraic*, and all the bases used in protocol must be hard-linear[9].

2.7 Cryptographic Hardness Assumptions

Here we state the hardness assumptions upon which the security of protocols that we analyze in this thesis rests. As mentioned earlier, models of security that we use here are computational: the running time and success or advantage probabilities of algorithms involved are modeled as functions of security parameter κ. Security, therefore, is only given for reasonable choice of the security parameter(s)[10].

In our work, we consider the hardness assumptions over prime order groups[11]. All the assumption that we use are commonly used in cryptography, except maybe Decisional Inverted-Additive Diffie-Hellman (DIDH) assumption. We will first present an assumption that is based on a computational problem (a search problem) within a cyclic group - Computational DiffieHellman assumption (CDH). Then we will present stronger, decisional discrete logarithm assumptions.

For a given security parameter κ, let $\mathbb{G}$ be a finite multiplicative group of prime order q, such that $|q| := \kappa$. Let g be a generator of $\mathbb{G}$. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in $[1 \dots q]$, and all operations on these are performed $\mod q$. We say that the assumption holds over $\mathbb{G}$ if there does not exist a (t, ε)-solver for polynomial t (in the security parameter κ governing the size of $\mathbb{G}$) and non-negligible ε.

2.7.1 Computational Assumptions

Computational Diffie-Hellman (CDH). Informally, we state CDH problem as follows: On input g, g^x, g^y, compute g^{xy}. Formally,

Definition 2.17. *Let $\mathcal{B}$ be a probabilistic algorithms trying to break a hardness assumption while running in time t and let $\varepsilon \in [0, 1]$. For $DH_g(g^x, g^y) := g^{xy}$, we say*

[9]To be more precise, Abdalla et al. in [1] use a weaker form of simulation-sound extractability, called algebraic-simulation-sound extractability. Further, the authors of [1] had to use algebraic model to achieve knowledge extractability property, since rewinding technique from [97] would induce an exponential blow-up in the security reduction of J-PAKE (and likewise of RO-J-PAKE and CRS-J-PAKE)

[10]When making a claim that something is hard with respect to the security parameter κ, this means that there exists no algorithm that can solve the problem in time that is polynomial in κ.

[11]As previously mentioned, the group of interest is either a Subgroup of Finite Fields (SFF) or Elliptic Curve (EC) group. Throughout this paper, protocols will be presented multiplicatively.

that $\mathcal{B}$ is a (t, ε)-CDH solver if

$$\mathbf{Succ}^{cdh}_{g,\mathbb{G}}(\mathcal{B}) := \Pr[\mathcal{B}(g, g^x, g^y) = DH_g(g^x, g^y)] \ \geq \ \varepsilon \ , \tag{2.8}$$

where x and y are chosen uniformly at random from $\mathbb{Z}_q$.

2.7.2 Decisional Assumptions

Since all decisional assumptions that we define here will have the same presentation format, we first provide a general formula and then define the specifics of each of the assumptions.

Definition 2.18. *For $* \in \{ddh, dtgdh, didh, dsdh\}$, let $\mathcal{C}$ be a challenger and let $\mathcal{D}$ be a probabilistic algorithms trying to break $*$ hardness assumption by playing the $*$-game against $\mathcal{C}$ while running in polynomial-time t and let $\varepsilon \in [0, 1]$. We say $\mathcal{D}$ is a (t, ε)-$*$ solver if*

$$\mathbf{Adv}^{*}_{g,\mathbb{G}}(\mathcal{D}) := \mathbf{Succ}^{*}_{g,\mathbb{G}}(\mathcal{D}) - \frac{1}{2} \geq \varepsilon, \tag{2.9}$$

where $\mathbf{Succ}^{*}_{g,\mathbb{G}}(\mathcal{D}) := \Pr[b' = b]$ in the $*$-game.

Decision Diffie-Hellman (DDH). Set $DH_g(g^x, g^y) := g^{xy}$, for any x, y and z from $\mathbb{Z}_q$. Set $* = \{ddh\}$. DDH-game is played as follows.

First, $\mathcal{C}$ flips a bit b, and chooses uniformly at random values x, y, and z in $\mathbb{Z}_q$. Then, $X := g^x$ and $Y := g^y$ are computed and Z is set as follows: $Z := g^z$ if b is equal to 0 and $Z := DH_g(X, Y)$ otherwise. Now, $\mathcal{D}$ gets as input (g, X, Y, Z) and tries to distinguish whether Z is the real Diffie-Hellman value $DH_g(X, Y)$ or a random group element of $\mathbb{G}$. At the end of the game, $\mathcal{D}$ outputs a bit b'.

Decision Triple Group Diffie-Hellman (DTGDH). For any x, y and z from $\mathbb{Z}_q$, set $DH_g(g^x, g^y) := g^{xy}$ and $TGDH_g(g^x, g^y, g^z) := g^{xyz}$. Set $* = \{dtgdh\}$. DTGDH-game is played as follows.

First, $\mathcal{C}$ chooses x, y, z, and w uniformly at random in $\mathbb{Z}_q$ and flips a bit b. $\mathcal{C}$ computes $X := g^x$, $Y := g^y$, and $Z := g^z$. The value W is set to g^w if $b = 0$, or $W := TGDH_g(X, Y, Z)$ otherwise. $\mathcal{D}$ gets (g, X, Y, Z, $DH_g(X, Y)$, $DH_g(X, Z)$, $DH_g(Y, Z)$, W), and tries to tell whether W is a Triple Diffie-Hellman value or a random group element. At the end of the game, $\mathcal{D}$ outputs bit a b'.

Decisional Inverted-Additive Diffie-Hellman (DIDH). For x and y in $\mathbb{Z}_q^*$, where $x + y \neq 0$, set $IDH_g(g^{1/x}, g^{1/y}) := g^{1/(x+y)}$. Set $* = \{didh\}$. DIDH-game is played as follows.

First, x, y, and z are chosen uniformly at random from $\mathbb{Z}_q^*$ and a bit b is flipped. Let $X := g^{1/x}$ and $Y := g^{1/y}$. If $b = 0$, $\mathcal{C}$ sets $Z := g^{1/z}$, and if $b = 1$, $Z := IDH_g(X, Y)$. $\mathcal{D}$ gets as input (g, X, Y, Z) and tries to tell whether Z is equal to $IDH_g(X, Y)$ value or it is equal to $Z := g^{1/z}$. At the end of the game, $\mathcal{D}$ outputs bit a b'.

Decision Square Diffie-Hellman (DSDH). For x from $\mathbb{Z}_q$, set $SDH_g(g^x) = g^{x^2}$. Also, set $* = \{dsdh\}$. DSDH-game is played as follows.

First x and y are chosen uniformly at random from $\mathbb{Z}_q$ and a bit b is flipped by $\mathcal{C}$. Let $X := g^x$. If the bit b that $\mathcal{C}$ holds is equal to 0, then $Y := g^y$. Otherwise, set $Y := SDH_g(X)$. Now, $\mathcal{D}$ gets as input (g, X, Y) and tries to distinguish whether Y is a square Diffie-Hellman value or a random group element of $\mathbb{G}$. At the end of the game, $\mathcal{D}$ outputs a bit b'.

2.7.3 Relation between Hardness Assumptions

When the reductionist approach is used, one needs to be aware that the proof of security typically does not provide absolute proof. Crucially, the scheme provides security guarantee relative to underlying *assumption*(s). As we said before, a precisely stated assumption is a prerequisite for the meaningful security analysis. When designing the cryptographic scheme and "choosing" underlying assumptions, preference is made to those assumptions that are well-studied and have stood the test of time. Also, a proof of security is more valuable when the assumption is weaker (i.e. it is implied by other, stronger assumption).

Now that we have introduced the comparable feature regarding security, we can compare security reduction results, i.e. if two schemes are proven secure, the one making weaker assumptions is preferable. However, it is not always possible to compare the strength of the assumptions. Luckily for us, all the assumptions that we use in this thesis are discrete logarithm based and therefore suited for comparison.

Comparison of discrete logarithm assumptions. In [1], the authors provide a comprehensive overview of known relations between different discrete logarithm based assumptions. In Figure 2.1, we will recall some of them.

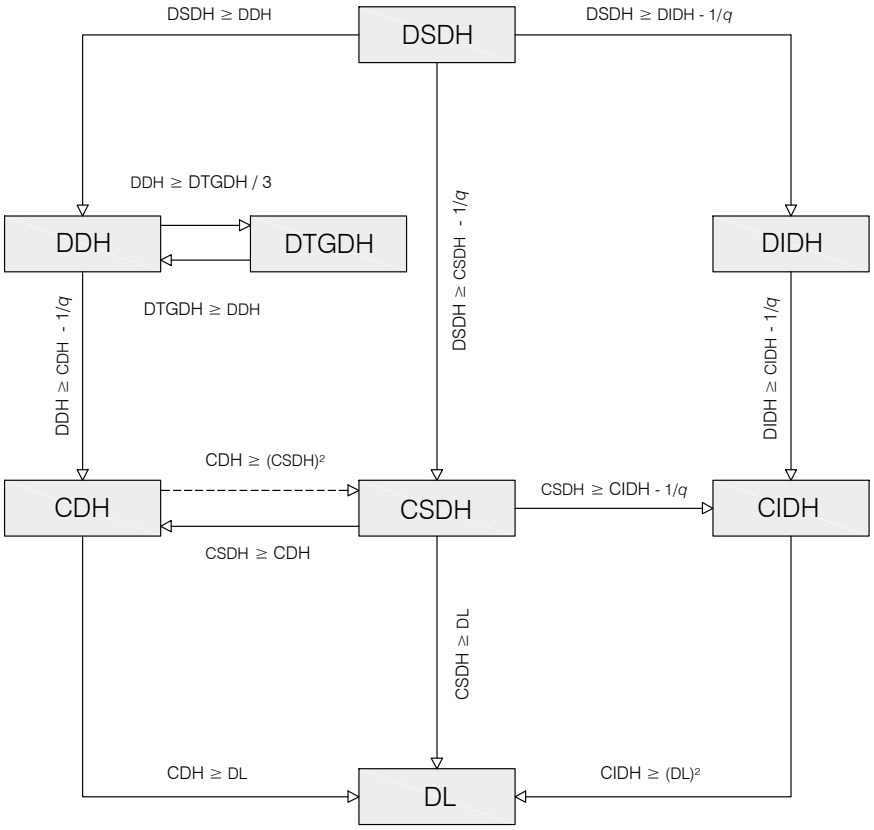

Figure 2.1: Relations between discrete logarithm assumptions

An arrow in Figure 2.1 will indicate an implication – if DDH is hard, then CDH must be hard, too. Thus, the assumptions on the top of the figure are believed to be stronger than those on the bottom; e.g. DSDH is belied to be the strongest assumptions out of all presented. From this figure we can also see that, in general, the assumptions that are based on a computational problem (i.e. CDH, CSDH, and CIDH) are believed to be weaker than their decisional counterparts (i.e. DDH, DSDH, and DIDH). We also have ordered assumptions by their "well-knownness". For instance, DIDH is less-known than DDH assumption. For the clarity of presentation, some of the known relations do not appear in Figure 2.1; e.g. [83] shows that DIDH is as hard as the Decisional Diffie-Hellman problem in generic groups.

Chapter 3

Password Authenticated Key Exchange

3.1 Introduction

Password Authenticated Key Exchange (PAKE) offers a cryptographic service that allows secure authenticated session key establishment over insecure networks between two or more parties who only share a low-entropy password. More specifically, involved parties, as a result of executing PAKE protocol, "boost" their low-entropy secret into a short-term, cryptographically-strong session key.

Loosely speaking, from the security perspective, we expect a PAKE protocol to be free of offline dictionary attacks against the user's password while limiting online password guessing attempts to a constant number per impersonation attempt. In other words, eavesdropping on PAKE communications leaks *no* password information to the adversary, and online interaction leaks the validity or invalidity of only a constant number (ideally, one) of password guesses[1].

3.1.1 Problem

Most PAKEs that are used in practice and appear in relevant standards – i.e. ISO [66], IETF [61], IEEE [65] – are studied using *game-based* models of security. Two most commonly used such models are Find-then-Guess (FtG) model of Bellare, Pointcheval, and Rogaway [9] and Real-or-Random (RoR) definition of Abdalla, Fouque and Pointcheval [3]. It is widely believed that these models render PAKE protocols that are well-balanced with regard to a trade-off between security they provide versus computational cost they require. However, the proof statements that are derived for

[1]Note that this is a purely *algorithmic* guarantee, independent of the implementation of a feature locking an account after too many failed login attempts.

PAKE protocols, after they have been analyzed in game-based models, suffer from several known limitations. Namely, the security analysis performed in such models typically considers PAKE protocol executed in protected environment, i.e., in isolation. As a consequence, the moment a PAKE protocol is composed with other protocols or primitives, and therefore input/output channels and the internal state of the global protocol principal are shared across multiple executions of different subprotocols and primitives, security guarantees that FtG and RoR model provide are not sufficient[2]. Another downside of such models is the common assumption that is being made on the distribution of the passwords: typically is it assumed that user passwords are selected independently from each other and are uniformly distributed over a password dictionary.

Furthermore, the internal state within these models is usually underspecified. Thus, in order for us to better understand composition properties of those PAKE protocols that are developed through the game-based framework, a slightly richer model of security is necessary; one which is able to represent input/output relations and local data in more detail.

3.1.2 Our Contribution

In this chapter, we present two well-established game-based models for PAKE, RoR and FtG, which we will use throughout this thesis. As our contribution, we provide a more detailed description of the internal state that is maintained during PAKE protocol execution within RoR security game. As it will become clear later in Section 4, this well-defined internal state will allow us to track more clearly an information flow between different algorithms when trying to establish composition guarantees for RoR secure PAKE protocols. We will also slightly modify the definition of acceptance and termination from [9], taking into account the fact that one should start using the session key only after the key exchange protocol is finished, and all checks are valid.

3.1.3 Previous Work

Key exchange research – even restricted to the case of passwords as a long-term authentication mechanism – is too vast to survey fully even for the purpose of this thesis. Hence, we shall briefly go over the landmark results for PAKEs, mostly focusing on papers that cover two party PAKEs.

[2]This, of course, does not mean that the security of the global protocol is automatically compromised in case of PAKE protocol composition or that the global protocol could not be secure in an adequate model of security.

PAKE, in general, has been very heavily studied in the past twenty-five years. Bellovin and Meritt pioneered the idea of PAKE in [13]. Their EKE protocol was the first of its kind to suggest that withstanding offline dictionary attacks was possible. Jablon's SPEKE protocol [67] soon followed with novel PAKE design insights. However, neither of these works contain formal security analysis. The first reasonable security models for PAKE appeared in [9] and [20] around the same time. While the latter (simulation based) has been used only in a few subsequent works [83, 87], it is the former (game-based) that has established itself as the model of choice when analyzing PAKEs.

Later, Katz et al. [71] showed how to *practically* realize provably secure PAKE without random oracles (but using a common reference string). This result has been used by Gennaro and Lindell in [46] to provide a general framework for PAKE in the common reference string model, making use of Smooth Projective Hash Functions (SPHF) from Cramer and Shoup [36]. In parallel to this, in more theoretical work Goldreich et al. [47] showed that PAKE is possible just using general complexity assumptions and no trusted setup whatsoever (i.e. in the plain model).

In [3], Abdalla et al. have shown that a stronger security notion (RoR) in the password case is needed if one aims for three-party PAKE[3]. The work that has contributed to secure composition of password-based key exchange with other protocols is one by Canetti et al. [29], in which authors have introduced the definition of *Universally Composable* (UC) PAKE. They also provided a construction based on the protocol from [71] that satisfies their UC definition. Finally, while all proposed PAKE protocols from previously mentioned works require a constant number of communication rounds to implement PAKE, Katz et al. [73] were first to propose a practical one-round PAKE.

The list of works that we covered is still incomplete; more relevant works on PAKE research can be found in Pointcheval's survey paper [96]. For a recent work that includes findings of more general field of password-based security and treats password registration problem, see Kiefer's thesis [74].

3.1.4 Organization

In Section 3.2, we formally define PAKE protocols and then mention those that are proven secure following the game-based models. Then, in Section 3.3 we first recall the FtG model from [9]. Then, we present extended Real-or-Random model (RoR) that

[3]In this 3-party PAKE setting, an honest-but-curious server plays as intermediary between number of clients that wish to communicate with each other, but only hold a single password.

was originally described in [3], making use of some notational elements from [26]. After describing two models for modeling PAKE, we introduce security properties which PAKE protocols should satisfy. Finally, we conclude the chapter in Section 3.4 with a discussion on the relation between two models.

3.2 PAKE Protocols

The objective of two-party password authenticated key exchange protocol is to allow two users who only share a low-entropy password to establish a high-entropy ephemeral session key despite the presence an unauthorized network-controlling adversary that does not posses the actual password. We formally define PAKE protocols as follows.

3.2.1 Definition

A PAKE protocol can be represented as a pair of algorithms $(PWGen, \mathrm{P})$: a password generation algorithm $PWGen$ and an algorithm P that defines the execution of the PAKE protocol. $PWGen$ takes as input a set of possible passwords **Pass**, equipped with a probability distribution $\mathcal{P}$. For simplicity of exposition, we make the assumption that $\mathcal{P}$ is the uniform distribution on **Pass**, and that user passwords are selected independently. We denote N the cardinality of **Pass**.

We can assume that P specifies several sub-algorithms, one of which is the initialization procedure that generates the system's public parameters common to all principals and initiates the internal state for each principal independently.

3.2.2 Real-World Protocols

As we mentioned before, there exists a plethora of PAKEs in the literature that were analyzed under *game-based* models of security for which our composition results from Section 4 may apply. Prominent examples include EKE [13], AuthA [12], PAK and PPK [84], SPAKE2 [4], KOY [71], the Denial-of-Service resistant OMDHKE [21], AugPAKE [107], TP-AMP [78], SNAPI [88], Dragonfly [79], J-PAKE [1] and its variants RO-J-PAKE and CRS-J-PAKE [80], Ring-Learning-with-Errors (RLWE)-based PAKE [39]. Note that SPEKE protocol from [67] has been proven secure by MacKenzie [83] in simulation-based model of [20]. Even though this model is claimed to be stronger than game-based models, there exists no formal proof of such claim. Nevertheless, the proof of Dragonfly from [79] (see also 6) that has been exhibited

using game-based definition provides a good indication that similar result should hold for SPEKE since the structure of these two protocols is very similar. Another relevant PAKE protocol that is included in ISO 11770-4 [66] and IEEE P1363.2 [65] standards is SRP from [116]. However, to date, no security proof has been provided in the literature for SRP protocol.

When considering relation between values that two communicating parties provide as input to the protocol, we distinguish between two forms of PAKEs: *balanced* PAKE also known as symmetric, in case of which two parties use the same value (password) to authenticate; and *augmented* PAKE (a.k.a verifier-based or asymmetric PAKE), which refers to a PAKE in which the server only stores a verifier of the actual password. With the exception of AugPAKE and TP-AMP, all the other cited protocol are balanced PAKE. However, for most of the mentioned protocols, an augmented version has been proposed.

3.3 Game-based PAKE Security Models

In this section, we present two game-based models that we will use in this thesis. First, we start by introducing the general setup and the security game mechanism that two models share. Second, we recall of Find-then-Guess (FtG) model from [9] that we will use in Section 6. Then, we will present extended Real-or-Random model (RoR) from [3], which will be used in Sections 4 and 5. Our variant of RoR model will additionally include a detailed description of the internal state in RoR security game. Also, minor changes are brought to the model in order to accommodate for forward secrecy. Finally, we introduce three standard security properties PAKE protocols are expected to satisfy together with the partner matching property which will make use of in Section 4.

Notation. Let us denote a game that represents †-security model of the PAKE for $\dagger \in \{FtG, RoR\}$ as $G^{\dagger}$. For such a game, there exist a challenger $\mathcal{CH}^{\dagger}$ that will keep the appropriate secret information away from an adversary $\mathcal{A}$ while administrating the security experiment.

Participants and passwords. In the two-party PAKE setting, each principal or user U is either from a $Clients$ set or a $Servers$ set, which are finite, disjoint, nonempty sets. The set ID_{pake} represents the union of $Clients$ and $Servers$. Furthermore, we assume that each client $C \in Clients$ possesses a password pw_C, while each

server $S \in Servers$ holds a vector of the passwords of all clients $pw_S := \langle pw_C \rangle_{C \in Clients}$. Following the convention in Section 3.2, these passwords are sampled independently and uniformly from **Pass** at the beginning of $G^{\dagger}$.

Protocol execution. The protocol P is a probabilistic algorithm that specifies the reaction of principals to network messages. In reality, each principal may run multiple executions of P with different users, thus in the model each principal is allowed an unlimited number of *instances* executing P in parallel. We denote U^i the i-th instance of principal U. In some places, where distinction matters, we will denote client instances C^i and server instances S^j.

When assessing the security of P, we assume that the adversary $\mathcal{A}$ has complete control of the network. In practice, this means that principals communicate solely through the adversary, who may consider delaying, reordering, modifying, and dropping messages sent by honest principals, or injecting messages of its choice in order to attack the protocol[4]. Moreover, $\mathcal{A}$ has access to principals' instances through the game's interface, which is offered by $\mathcal{CH}^{\dagger}$.

3.3.1 The Find-then-Guess Model

In FtG model, while playing the security game, $\mathcal{A}$ provides the inputs to $\mathcal{CH}^{FtG}$ – who parses the received messages and forwards them to corresponding instances – via the following *queries*:

- **Send**(U^i, M): $\mathcal{A}$ sends message M to instance U^i. As a response, U^i processes M according to P, corresponding internal state is updated, and instance outputs a reply that is given to $\mathcal{A}$. A **Send**(U^i, **Start**) has instance U^i output P's first message. **Send** query models active attacks.

- **Execute**(C^i, S^j): This triggers an honest run of P between client C^i and server S^j, and the transcript of the protocol execution is given to $\mathcal{A}$. It covers passive eavesdropping on protocol flows.

- **Reveal**(U^i): As a response to this query, $\mathcal{A}$ receives the current value of the session key sk_U^i. $\mathcal{A}$ may do this only if U^i holds a session key and a **Test** query has not been made to U^i or its partner. This query captures potential session key leakage as a result of its use in higher level protocols. Also, it ensures that if some session key gets exposed, other session keys remain protected.

[4]This model assumes that the passwords setup procedure is private (secure).

- **Test**(U^i): Upon receiving this query, a bit b is flipped by $\mathcal{CH}^{FtG}$. If $b = 1$, $\mathcal{A}$ gets sk_U^i. Otherwise, it receives a random string from the session key space. Note that only a *fresh* instance can be a target of a **Test** query. Restriction in this model is that at any time during the execution $\mathcal{A}$ may only make one such query. This query measures sk_U^i's semantic security.

- **Corrupt**$(U, (pw_{\mathcal{A}}, V))$: The password pw_U is given to $\mathcal{A}$ if U is a client, and the list of passwords pw_U in case U is a server. Optionally, the adversary may replace the value of password from client U with the value $pw_{\mathcal{A}}$ in the list of passwords used by server V. This models compromise of the long-term key and captures forward secrecy notion (see below)[5].

As can be seen above, the adversary is allowed to send multiple **Send**, **Execute**, **Reveal** and **Corrupt** queries to the challenger, and only a single **Test** query. Note that the validity and format of each query are checked upon receipt.

Accepting and terminating. In the FtG model from [9], an instance U^i accepts ($acc_U^i := 1$) if it holds a session key sk_U^i, a session ID sid_U^i and a partner ID pid_U^i. An instance U^i terminates ($term_U^i := 1$) if it will not send nor receive any more messages. U^i may accept and terminate *once*.

Partnering. We say that instances C^i and S^j are partnered if: (1) $acc_C^i = 1$ and $acc_S^j = 1$; (2) $sid_C^i = sid_S^j \neq \perp$; (3) $pid_C^i = S$ and $pid_S^j = C$; (4) $sk_C^i = sk_S^j$; and (5) no other instance accepts with the same sid.

Freshness. Freshness captures the idea that the adversary should not trivially know the session key being tested. An instance U^i is said to be fresh (without forward secrecy) if: (1) the instance has computed and accepted a session key ($acc_U^i := 1$); (2) no **Reveal** query was made to U^i nor to its partner U'^j (if it has one); (3) no **Corrupt** query has been made since the beginning of the game. If we consider forward secrecy, an instance is said to be fs-fresh if rules (1) and (2) from above definition are satisfied and if (3) no **Corrupt**(U') query was made before the **Test** query and a **Send**(U^i, M) query was made at some point, where U' is any participant.

[5]This is the weak-corruption model of [9].

3.3.2 The Real-Or-Random Model

In RoR model, while playing the security game, $\mathcal{A}$ provides the inputs to $\mathcal{CH}^{RoR}$ – who parses the received messages and forwards them to corresponding instances – via the following *queries*:

- **Send**(U^i, M): $\mathcal{A}$ sends message M to instance U^i. As a response, U^i processes M according to P, the corresponding internal state (see below) is updated, and the instance outputs a reply that is given to $\mathcal{A}$. A **Send**(U^i, V) query has instance U^i output P's first message, destined to principal V. The purpose of the **Send** query is to model active attacks.

- **Execute**(C^i, S^j): This query triggers an honest run of P between client C^i and server S^j, and the transcript of the protocol execution is given to $\mathcal{A}$. It covers passive eavesdropping on protocol flows[6].

- **Reveal**(U^i): As a response to this query, $\mathcal{A}$ receives the current value of the session key $sk\mathrm{P}_U^i$. $\mathcal{A}$ may do this only if U^i holds a session key (i.e., has accepted, see below) and a **Test** query has not been made to U^i or its partner instance (see below). This query captures potential session key leakage as a result of its use in higher level protocols. It ensures that if some session key gets exposed, other session keys remain protected.

- **Test**(U^i): A hidden bit b is randomly selected by $\mathcal{CH}^{RoR}$ at the very beginning of G^{RoR} and the same bit is used for all **Test** queries. If $b = 1$, $\mathcal{A}$ receives $sk\mathrm{P}_U^i$ as an answer to the **Test**(U^i) query. Otherwise, $\mathcal{A}$ receives a random string from the session key space[7]. Note that in contrast to the FtG model, some additional care must be taken in case the **Test** keys are random ($b = 0$): $\mathcal{CH}^{RoR}$ needs to make sure that two *partnered* instances will respond with the same random value. Note that only a *fresh* instance can be a target of a **Test** query. This query measures the semantic security of session keys.

- **Corrupt**(U): The password pw_U is given to $\mathcal{A}$ if U is a client, and the list of passwords pw_U in case U is a server. This query models compromise of the long-term key and captures forward secrecy (see below).

[6]Note that **Send** and **Execute** queries are exactly the same as in FtG model.

[7]Thus, the session keys that are forwarded to $\mathcal{A}$ in response to **Test** queries are either all real or all random.

As can be seen above, the adversary is allowed to send multiple **Send**, **Execute**, **Reveal**, **Corrupt**, and **Test** queries to the challenger. Note that the validity and format of each query are checked upon receipt.

Internal state and initialization. In the interest of running a sound simulation, the challenger $\mathcal{CH}^{RoR}$ maintains two types of *internal state* in the form of certain random variables, and updates it as (1) the actual network interactions between $\mathcal{A}$ with the instances running P on the lower level and (2) the security game G^{RoR} on the higher level, progress. The first type of internal state contains the necessary data for the actual execution of P by the instances. It is called the *execution state* EST_{pake}. The another kind of state is called the *game state* GST_{pake}; it stores information used by $\mathcal{CH}^{RoR}$ to keep track of and administer the game, as well as define security (e.g. a hidden bit, flags that indicate corruptions, etc). Figure 3.1 lists all of these variables, which we also detail below. Notice that variables may be user-specific (e.g. passwords), others are defined per instance (e.g. $status\mathrm{P}_U^i$, sid_U^i, $sk\mathrm{P}_U^i$), and yet others are global to the game (e.g. the **Test** bit).

Execution state. Let U be a user and U^i an instance. U holds – and instance U^i uses – the long-term keying information pw_U, set at the beginning of the game. The session identifier sid_U^i uniquely labels the protocol session U^i wishes to establish with some other instance. It starts out set to $\perp$. U^i holds a partner identifier $pid\mathrm{P}_U^i$, representing the identity of the principal with which U^i believes it shares a session key. It also starts out set to $\perp$. The value $status\mathrm{P}_U^i$ tracks the status of an instance U^i, i.e whether it is *running* (waiting for the next protocol message), has *accepted* (thus holds a session key), or has *rejected*. It is initially set to *running*. A variable $info\mathrm{P}_U^i$ stores any additional information U^i needs in order to perform its computations (e.g. exponents for Diffie-Hellman-type terms). Finally, the variable $sk\mathrm{P}_U^i$, which starts out at $\perp$ holds the session key which U^i may establish during a protocol run.

Game state. Concerning the game state, we first have the *test bit* b which is flipped by $\mathcal{CH}^{RoR}$ at the beginning of the game. The flag r_U^i indicates the *status of a session key* held by U^i, thus showing if the instance has been a target of a **Reveal** query or not. It is set to $\perp$ until $sk\mathrm{P}_U^i$ is non-$\perp$; then it is by default set to *hidden*. Similarly, t_U^i shows if an instance has been the target of a **Test** query. It is set to $\perp$ until $sk\mathrm{P}_U^i$ is non-$\perp$; then it is by default set to *untested*. The variable $pnr\mathrm{P}_U^i$, which starts out as $\perp$, stores the identity of the partner *instance*. *Freshness* of an instance

(defined below) is tracked with f_U^i; this is set to $\perp$ until $sk\mathrm{P}_U^i$ is non-$\perp$. Then, it is set by default to $fresh$. Lastly, the corruption flags are maintained: (1) the value δ indicates if a **Corrupt** query has been made so far; (2) the flag δ_U^i, which starts out set to *honest*, shows whether the instance U^i received a message after some password disclosure has occurred[8]: the value of the flag switches from *honest* to *corrupted* only if a **Send**(U^i, M) query was made while $\delta = 1$ and $status\mathrm{P}_U^i = running$. Note that in particular, if U^i is the target of an **Execute** query while $\delta = 1$, then δ_U^i remains set to *honest*.

Internal State: For $U \in ID_{pake}$, $C \in Clients$, $S \in Servers$, and $i \in \mathbb{N}$, the internal state is maintained as follows:

- **Execution State** EST_{pake}
  - $\star$ $pw_C \in$ **Pass**; $\quad pw_S := \langle pw_C \rangle_C$
  - $\star$ $status\mathrm{P}_U^i \in \{running, accepted, rejected\}$
  - $\star$ $sid_U^i,\ sk\mathrm{P}_U^i,\ info\mathrm{P}_U^i \in \{0,1\}^* \cup \{\perp\}$; $\quad pid\mathrm{P}_U^i \in ID_{pake} \cup \{\perp\}$
- **Game State** GST_{pake}
  - $\star$ $b \in \{0,1\}$; $\quad r_U^i \in \{\perp, hidden, revealed\}$; $\quad t_U^i \in \{\perp, untested, tested\}$
  - $\star$ $pnr\mathrm{P}_U^i \in ID_{pake} \times \mathbb{N} \cup \{\perp\}$; $\quad f_U^i \in \{\perp, unfresh, fresh\}$
  - $\star$ $\delta \in \{0,1\}$; $\quad \delta_U^i \in \{honest, corrupted\}$.

Figure 3.1: The internal state of PAKE in RoR model

Initialization. In an initialization phase (see Fig. 3.2), which occurs before the execution of a protocol, public parameters and the internal state are fixed. The appropriate sub-algorithm of P, called $ParamGen$, is run to generate the system's public parameters $Params\mathrm{P}$. From the adversary's perspective, an instance comes into being after **Send**$(U^i,$ **Start**$)$ query with the **Start** is asked. For each client a secret pw_C is drawn uniformly (and independently) at random from a finite set $Pass$ of size N and is given to all servers.

[8]Note that here we mean that any **Corrupt** query may have been asked and not necessary one that targets the principal U.

InitialPAKE(1^k): The *Clients* and *Servers* set are fixed in advance. For $U \in ID_{pake}$, $C \in Clients$, $S \in Servers$ and $i \in \mathbb{N}$, the initialization procedure is performed as follows:

- **Generate** $Params\mathrm{P}$
  - $\star$ $Params\mathrm{P} \leftarrow ParamsGen(1^k)$
- **Initialize** EST_{pake}
  - $\star$ $pw_C \leftarrow$ **Pass**; $\quad pw_S[C] := pw_C$
  - $\star$ $status\mathrm{P}_U^i := running$
  - $\star$ $sid_U^i,\ sk\mathrm{P}_U^i,\ info\mathrm{P}_U^i,\ pid\mathrm{P}_U^i := \perp$
- **Initialize** GST_{pake}
  - $\star$ $b \leftarrow \{0,1\}$; $\quad r_U^i,\ t_U^i := \perp$
  - $\star$ $pnr\mathrm{P}_U^i,\ f_U^i := \perp$
  - $\star$ $\delta := 0$; $\quad \delta_U^i := honest$
- **Return** $Params\mathrm{P}$.

Figure 3.2: The initialization procedure for PAKE

Accepting, rejecting, and continuing. Since in practice one should start using the session key only after the key exchange protocol is finished, we took the definition from the FtG model (see Section 3.3.1) and collapsed "terminating" into "accepting". As a result, we started tracking this event with the variable $status\mathrm{P}_U^i$ which in addition to *running* and *rejected*, can take value *accepted*. The variable $status\mathrm{P}_U^i$ is changed to *accepted* once an instance U^i sets $sk\mathrm{P}_U^i$ to a non-$\perp$ value. It then no longer accepts or sends any messages. This change brings only minor alterations to standard partnering and freshness definitions. We require that if U^i accepts, then sid_U^i is non-$\perp$, and we also require that if $sid_U^i \neq \perp$, then $pid\mathrm{P}_U^i \neq \perp$. However, it can be that $sid_U^i \neq \perp$ and $sk\mathrm{P}_U^i = \perp$.

When $status\mathrm{P}_U^i = running$, the instance has neither accepted a session key nor rejected. It simply is waiting for the next protocol message. When $status\mathrm{P}_U^i = rejected$, it stops sending and receiving messages and refuses to generate a session key. Note that an instance can very well reject even if $sid_U^i \neq \perp$.

Partnering and semi-partnering. It is necessary to distinguish partnering from semi-partnering in order to properly deal with authentication (see below). Specifically, semi-partnering is used to bind an instance that has accepted to the instance that could become its partner *if the last protocol message is delivered*. Of course, this delivery is entirely up to $\mathcal{A}$.

Semi-partnering. We say that instance U^i is a *semi-partner instance* to S^j and vice versa if: (1) U is a client and V is a server or vice versa, (2) $sid := sid_U^i = sid_V^j \neq \perp$, (3) $pid\mathrm{P}_U^i = V$ and $pid\mathrm{P}_V^j = U$, and (4) no other instance has a non-$\perp$ session identity equal to sid.

Partnering. We say that instance U^i is a *partner instance* to V^j and vice versa if: (1) U^i is a semi-partner to V^j, (2) $status\mathrm{P}_U^i = status\mathrm{P}_V^j = accepted$, and (3) $sk\mathrm{P}_U^i = sk\mathrm{P}_V^j$.

Freshness. *Freshness* captures the idea that the adversary should not trivially know sessions keys being tested. Recall that by default, an instance *that has accepted* has the flag f_U^i set by default to $fresh$. (Before acceptance, it is $\perp$.) This value switches to $unfresh$ if any of the following conditions holds: (1) $r_U^i = revealed$, or (2) if U^i has a partner instance V^j and $r_V^j = revealed$, or (3) $\delta_U^i = corrupted$.

An instance is said to be *fresh* if it has accepted and $f_U^i = fresh$.

3.3.3 Security Properties

Now that we have defined accepting, partnering, freshness and all the queries available to the adversary $\mathcal{A}$, we can formally define the authenticated key exchange (ake) advantage of $\mathcal{A}$ against P, authentication properties, forward secrecy and partner matching for both security games, G^{RoR} and G^{FtG}.

AKE security. Eventually, $\mathcal{A}$ ends the game and outputs a bit b'. We say that $\mathcal{A}$ wins and breaks the ake security of P if $b' = b$, where b is the hidden bit selected at the beginning of the protocol execution in G^{RoR}, or once the **Test** query was asked in G^{FtG}. We denote the probability of this event by $\mathbb{P}[b' = b]$. In both security games, G^{RoR} and G^{FtG}, the *ake*-advantage of $\mathcal{A}$ in breaking P is usually defined as

$$\mathbf{Adv}_{\mathrm{P}}^{ake}(\mathcal{A}) := 2 \cdot \mathbb{P}[b' = b] - 1. \tag{3.1}$$

It turns out to be more convenient for us later to reformulate the *ake*-advantage function. Let b be a bit. For $\dagger \in \{FtG, RoR\}$, we denote $G^{\dagger - b}$ the game played exactly as $G^{\dagger}$, except that (1) the bit b has been fixed in advance and (2) at the end

of the game $G^{\dagger-b}$ outputs a final bit denoted b'' computed as follows: $b'' := 1$ if and only if $b = b'$. (Recall that b' is the bit output by $\mathcal{A}$.) Using $\mathbb{P}^b$ to denote probabilities in the space defined by game $G^{\dagger-b}$, it is then easy to see that $\mathbf{Adv}^{ake}_{\mathrm{P}}(\mathcal{A})$ as defined in Eq. 3.1 can be re-written as

$$\mathbf{Adv}^{ake}_{\mathrm{P}}(\mathcal{A}) := \mathbb{P}^1[b'' = 1] - \mathbb{P}^0[b'' = 0]. \tag{3.2}$$

Finally, we say that P is *ake*-secure if there exists a positive constant B such that for every probabilistic, polynomial-time adversary $\mathcal{A}$ it holds that

$$\mathbf{Adv}^{ake}_{\mathrm{P}}(\mathcal{A}) \leq \frac{B \cdot n_{se}}{N} + \varepsilon \tag{3.3}$$

where n_{se} is an upper bond on the number of **Send** queries $\mathcal{A}$ makes, and ε is negligible in the security parameter. Recall that N is the cardinality of **Pass** and that passwords are assigned uniformly at random to users. This formula adequately captures the idea that an adversary's advantage in breaking a PAKE should only significantly grow if the adversary actively tests candidate passwords against user instances. In particular, offline dictionary attacks are completely ruled out.

Authentication. Another of $\mathcal{A}$'s goals is violating authentication. As in [9], we define three notions of authentication: *client-to-server*, *server-to-client*, and *mutual*. We denote by **c2s** (respectively, **s2c**) the event that some server (resp., client) instance has accepted before any **Corrupt** query without being *semi-partnered* to a client (resp., server) instance. The adversary is said to violate mutual authentication if there exists some instance that has accepted before any **Corrupt** query without a semi-partner. We denote by **ma** this event. The *c2s*-advantage, *s2c*-advantage, and *ma*-advantage of $\mathcal{A}$ in breaking P are respectively

$$\mathbf{Adv}^{c2s}_{\mathrm{P}}(\mathcal{A}) := \mathbb{P}[\mathbf{c2s}], \quad \mathbf{Adv}^{s2c}_{\mathrm{P}}(\mathcal{A}) := \mathbb{P}[\mathbf{s2c}], \quad \text{and} \quad \mathbf{Adv}^{ma}_{\mathrm{P}}(\mathcal{A}) := \mathbb{P}[\mathbf{ma}]. \tag{3.4}$$

We say that the protocol P is $\star$-secure for $\star \in \{c2s, s2c, ma\}$ if for every efficient adversary $\mathcal{A}$, $\mathcal{A}$'s advantage function $\mathbf{Adv}^{\star}_{\mathrm{P}}(\mathcal{A})$ is upper bounded as in Eq. 3.3.

Forward Secrecy. While being considered an advanced security feature, Forward Secrecy (FS) is a valuable property in the context of key exchange protocols. It provides the guarantee that past session keys will not be automatically divulged by the compromise of long-term keys (obviously assuming that past session keys are deleted from memory before the compromise).

In game-based models for key exchange protocols, the notion of forward secrecy is captured by allowing the adversary to make a **Corrupt** query, which may come in different flavors. Namely, several types of **Corrupt** queries have appeared in the literature, based on the amount of secret information the adversary learned or had the opportunity to modify. We present them in Fig. 3.3.

Full disclosure: The adversary may learn all information in the memory of the targeted user. This includes the long-term secret key of the user, as well as all the internal state values of the execution state (of all instances) and session keys stored in the memory.

Partial disclosure: The adversary may learn only partial information from the memory of the targeted user. In this case the adversary may ask two types of **Corrupt** queries: $\textbf{Corrupt}_1(U)$ which leaks some secret information of user U and $\textbf{Corrupt}_2(C, S, pw)$ which allow the adversary to modify information stored in user's (typically server's) memory. In the password setting, based on the types of queries that are available to the adversary and their strength we distinguish between following cases:

1. only $\textbf{Corrupt}_1(C)$: the targeted client C returns his password pw, while servers are not vulnerable to this type of query [1],
2. both $\textbf{Corrupt}_1(C)$ and $\textbf{Corrupt}_2(C, S, pw)$ are possible: the targeted client C returns his password pw, while the targeted server S may allow the adversary to modify passwords stored on S without learning the value of the old password [72],
3. only $\textbf{Corrupt}_1(U)$: the targeted client C returns his password pw, while the targeted server returns the list of passwords $pw_S[C]$ [84],
4. both $\textbf{Corrupt}_1(U)$ and $\textbf{Corrupt}_2(C, S, pw)$ are possible: disclosed information from previous case (3) + the adversary is allowed to modify passwords stored in the server's memory [9].

Figure 3.3: Different types of Corrupt query in PAKE

The strength of FS property can be additionally fine-tuned through the freshness definition or the power given to the adversary. For the latter, imagine a (weak) type of FS which is guaranteed for any session key that is established without the active intervention of the attacker (except for eavesdropping on communications). In that case, the session key is forward secure as soon as it is erased from memory. The HMQV protocol, for instance, satisfies this type of FS.

Partner Matching Here, we define *partner matching*, which is an additional property PAKE protocol may need to satisfy. Informally, we say that a PAKE protocol satisfies *partner matching* if an observer of network communications can deduce which instances are partnered according to our definition.

More formally, we define an efficient partner matching algorithm M that enables the observer of the security game (either FtG or RoR) to identify partner instances by outputting a partnering list $\mathcal{L}_{pnr\mathrm{P}}$ that consists of pairs of user instances (U^i, V^j), one of which is marked as $\perp$ if an instance does not have a partner yet. The input to the algorithm M includes all the queries the challenger receives from the adversary $\mathcal{A}$ and all the replies the challenger returns to $\mathcal{A}$ together with all the public values.

Note that in the FtG model of [9], the session identifier *sid* is given to the adversary after instance accepts and therefore is a public information. In [26], this is claimed to be enough to assume that a partner matching algorithm is available. This happens to be often true in the PAKE case. Indeed, most PAKEs in the literature rely on session identifiers built as concatenations of sent and received messages and identities to determine partners. This makes partner matching immediate.

3.4 Model Comparison

In this section we discuss about the relation between existing models: First we compare FtG model [9] with the original RoR model [3]. Then, we explain the changes we made to the original RoR model to allow the treatment of forward secrecy. Finally, we show how this changes influence the relation between FtG and our RoR.

3.4.1 FtG vs RoR Model

As could be seen so far, the Real-Or-Random model (RoR) is very close to the Find-Then-Guess model (FtG). The main difference between them is that the RoR model allows the adversary to send multiple **Test** queries to *different* fresh instances, instead of a single **Test** query that is available in the FtG security game. Consequently, several modifications are brought to RoR in relation to the **Test** query handling. For example, a hidden bit b is randomly selected by a challenger $\mathcal{CH}^{RoR}$ at the very beginning of an RoR game and the same bit is used for all received **Test** queries. Another divergence of the two models is around the **Reveal** query. In the original RoR model [3], the **Reveal** query is disallowed, while misuse of the keys (session key leakage) is modeled with the **Test** queries. In both models, FtG and RoR, we say that two party PAKE protocol P is secure in ake sense if the advantage of an adversary is only negligible

larger that $\frac{B \cdot n_{se}}{N}$, where B is some constant, n_{se} counts as the adversary's online attacks and N is the size of the dictionary.

When considering relation among two game-based models of security for PAKE, Abdalla, Fouque, and Pointcheval in [3] have proven that the RoR security implies the FtG security by showing that a security proof in the RoR model can be translated into one in the FtG model with loss in the reduction of only a factor of 2. For the other direction, they shown that there exists a loss of a non-constant factor in the reduction, that cannot be avoided, which in the case of password-based key exchange is troublesome.

3.4.2 Original RoR vs Our RoR Model

Recall that the **Reveal** query was disallowed in the original RoR model [3]. There, misuse of the keys (session key leakage) was modeled solely using **Test** queries. Nevertheless, as in the recent works [1] and [80], in our RoR model we again allow the adversary to make the **Reveal** query. This change is essential to accommodate corruption queries in the RoR model. More specifically, the reason for re-inclusion of the **Reveal** query is the following. After the corruption of any principal, the adversary is no longer allowed to make **Test** queries to unfresh instances. Therefore, the **Reveal** query is needed to model misuse of those keys as well.

3.4.3 FtG vs Our RoR Model

Now, let us consider the RoR model with the **Reveal** query as in our RoR model presented above and in [1, 80]. This means that in the security reduction from [3] (Lemma 1), $\mathcal{A}^{RoR}$ is allowed to ask the **Reveal** query, which will in turn allow perfect simulation of $\mathcal{A}^{FtG}$ so factor 2 can be dropped ($\mathcal{A}^{RoR}$ does not have to pick his bit in the reduction anymore, only $\mathcal{CH}^{RoR}$). In the other direction, we still have the same non-preserving security reduction. Therefore, in case a **Reveal** query is allowed in the RoR model, the RoR security model is strictly stronger than the FtG security model.

Forward secrecy comparison. Regarding forward secrecy, a **Corrupt** query in one model can be simulated with a **Corrupt** query in the other (assuming they are of same strength according to Figure 3.3). Since no significant change is made in the RoR model, neither to the definition of freshness nor the definition of partnership, with respect to forward secrecy, the two models are equivalent.

However, in combination with multiple **Test** queries certain differences in the protocol security analysis may arise. Namely, sometimes due to the **Corrupt** query in AKE proofs, for the reduction to work, the challenger needs to guess the *target* instance – the instance that is the target of the **Test** query (FTG model) – in which to plant the hardness assumption challenge. As a consequence, the quality of the reduction is affected: degradation of the security in the number of initialized instances occurs. This security degradation is more hurtful in the FtG model than in RoR, where the challenger $\mathcal{CH}^{RoR}$(at the same time the adversary against the hardness assumption) has plenty of target instances to choose from, due to multiple **Test** queries.

Chapter 4

Composability of Game-based PAKE Protocols

4.1 Introduction

Even though there may be other applications of PAKE, it is common practice that the secret key derived from a PAKE execution is used to authenticate and encrypt some data payload using symmetric key primitives and protocols. For example, two certificate-less TLS proposals that integrate PAKE as a key exchange mechanism have recently appeared on the IETF [35, 58][1]. When looking at these two proposals through the lense of composition, one sees that both of them suggest the PAKE be followed by Authenticated Encryption (AE) algorithms (namely AES-CCM and AES-GCM). Another project that makes use of PAKE is Magic Wormhole [114], the file transfer protocol in which PAKE is composed with NaCl's *crypto_secretbox* containing the stream cipher XSalsa20 and MAC algorithm Poly1305. Consequently, being able to guarantee the overall security of a *composed* protocol, consisting of first running a PAKE and then a symmetric key application, is very important.

4.1.1 Problem

Unfortunately, a provably secure composition is difficult to automatically obtain without using complex, usually simulation-based models. Furthermore, most PAKEs in the literature are studied using so-called *game-based* models, which – while being workable in order to obtain acceptable proofs – do not guarantee secure composition. The most common such model used is the Find-then-Guess (FtG) model of Bellare

[1]The reason behind this integration - and not using PAKE with some symmetric cipher over TCP - is to circumvent the need to establish a network protocol for data transfer (i.e. TCP or UDP) and to negotiate symmetric key algorithms (or protocols) on their own.

et al. [9]. However, up until now, the only known composition guarantees provided by this model are of the weakest type: the concurrent self-composition.

The composition result for game-based AKE. In [26], Brzuska et al. show that a middle ground is possible in the case *Public-Key Infrastructure*-based key exchange (PKI-KE) that is secure in the FtG model from [10], which is the AKE model analog to one from [9] that covers PAKEs. Among other things, they define a framework for PKI-KE that (1) is game-based and (2) allows to prove that, if a public matching algorithm exists (see Section 3.3.3), secure composition holds when the class of higher-level applications is restricted to symmetric-key protocols. To develop this further, we briefly explain the theorem of Brzuska et al. [26] and then show where passwords cause trouble.

Let S be some arbitrary, two-party, symmetric key protocol and P; S denote its "natural" composition with P. The main theorem established in [26] for the PKI-KE case states that for every efficient adversary $\mathcal{A}$ playing a suitably defined security game against P; S there exist efficient adversaries $\mathcal{B}$ against P and $\mathcal{C}$ against S such that following formula holds:

$$\mathbf{Adv}_{\mathrm{P;S}}(\mathcal{A}) \leq q \cdot \mathbf{Adv}_{\mathrm{P}}^{FtG}(\mathcal{B}) + \mathbf{Adv}_{\mathrm{S}}(\mathcal{C}), \tag{4.1}$$

where q is the maximum number of instances in play in the key exchange game. Of course, in [26]'s framework, security of the composition holds when the left-hand term is negligible. Therefore, the upper bound implies this under the condition that P and S are secure. Indeed, observe that q is at most polynomial in the security parameter and that $\mathbf{Adv}_{\mathrm{P}}^{FtG}(\mathcal{B})$ is supposed to be *negligible* when using PKI-KE. (And, of course, S is secure if $\mathbf{Adv}_{\mathrm{S}}(\mathcal{C})$ is negligible for all $\mathcal{C}$.) This effectively shows that the security of the composition P; S is guaranteed by the stand-alone security of P and S. Let us now examine whether their result carries over to PAKE setting.

Two immediate password problems. It is well-known that already when dealing with "basic" PAKE definitions, the usual low-entropy nature of the long-term authentication material causes definitional headaches. It is, therefore, no surprise that similar issues should be encountered here.

There are two main obstacles to overcome when trying to get a password analog of Eq. 4.1 to work, and both stem from the non-negligible term in Eq. 3.3. First, it is clear that the term $q \cdot \mathbf{Adv}_{\mathrm{P}}^{FtG}(\mathcal{B})$ cannot be negligible anymore. Thus, it makes no sense to try and deduce from Eq. 4.1 that the left-hand side is ultimately negligible.

The only way out of this is to "boost" the left-hand side. Fortunately, there is a natural way to do this. Indeed, intuitively it should be clear that the composed protocol will also suffer from a breach in the event $\mathcal{A}$ guesses a password and mounts an online attack. Thus, it is the definition of security for the composed protocol that has to change, in that it needs to incorporate the same non-negligible bound as in Eq. 3.3. In other words, at best we can only require by definition that:

$$\mathbf{Adv}_{\mathrm{P;S}}(\mathcal{A}) \leq \frac{B \cdot n_{se}}{N} + \varepsilon, \tag{4.2}$$

where B is some constant, ε is negligible, and n_{se} counts $\mathcal{A}$'s online attacks. In short, our first problem is handled at the definition level. But, Eq. 4.2 leads into our second problem.

If we simply plug our optimal FtG PAKE bound into the right-hand side of Eq. 4.1, we obtain

$$\mathbf{Adv}_{\mathrm{P;S}}(\mathcal{A}) \leq \frac{B \cdot q \cdot n_{se}}{N} + \mathbf{Adv}_{\mathrm{S}}(\mathcal{C}). \tag{4.3}$$

This is not what we want: The q factor is still making the desired upper bound too large for our purpose! Therefore, an open question that we tackle in this chapter is whether or not a similar result to one from [26] can be exhibited for PAKE.

4.1.2 Our Contribution

We answer this question positively by essentially adapting the framework in [26] to the password-based case. Namely, in the proof of the main theorem in [26], the authors need to make use of a hybrid argument indexed by the instances in play: The idea is to have the simulator plant the only available **Test** query at the randomly chosen index. This is what makes the q come out. Our observation is that by using the RoR model – in which *multiple* **Test** queries are allowed – we can avoid having this parasite factor appear.

In short, our main theorem says that for every efficient adversary $\mathcal{A}$ playing against P; S there exist efficient adversaries $\mathcal{B}$ and $\mathcal{C}$ such that:

$$\mathbf{Adv}_{\mathrm{P;S}}(\mathcal{A}) \leq \mathbf{Adv}^{RoR}_{\mathrm{P}}(\mathcal{B}) + \mathbf{Adv}_{\mathrm{S}}(\mathcal{C}), \tag{4.4}$$

and from this theorem we get that if P and S are actually secure, the optimal bound stated in Eq. 4.2 holds[2].

[2]Note that the presence of passwords has no effect on the security of S as a stand-alone primitive. This is why $\mathbf{Adv}_{\mathrm{S}}(\mathcal{C})$ should remain negligible.

More concretely, we show that PAKE protocols secure in the sense of the game-based Real-or-Random (RoR) definition of Abdalla et al. [3] allow for automatic, secure composition with arbitrary, higher-level symmetric key protocols according to a security definition very similar to that in [26]. Since in [3] the authors provide evidence that most PAKEs secure in the FtG model of [9] are in fact secure according to RoR, we can conclude that nearly all provably secure PAKEs enjoy a certain degree of composition, one that at least covers the case of implementing secure channels. It should be noted that for our result to hold, we also need the public matching algorithm mentioned earlier to be fulfilled. However, we emphasize that to the best of our knowledge, for nearly all published PAKEs this is always the case. Prominent examples include EKE [13], PAK [84], SPAKE2 [4], Dragonfly [79], and J-PAKE [1] and its versions [80].

4.1.3 Previous Work

From a strict PAKE standpoint, the work most relevant to ours is [3], where it was shown among other things that allowing multiple Test queries in the model – as opposed to only one (as in [9]) – yields a strictly stronger security notion in the password case.

Composition of game-based key exchange. All key exchange models devised in the last two decades (e.g. [10, 109, 9, 20, 30, 3]) support concurrent self-composition of key exchange protocols[3]. Although key exchange protocols proven in the game-based setting of Bellare and Rogaway (BR) [10] remained mostly used in practice, it took almost a decade before someone started addressing the problem of studying the composability properties of this setting. The first to successfully provide a framework in the game-based setting that grants stronger composition guarantees were Canetti and Krawczyk [30]. Indeed, they identified a security notion (SK-security) that is sufficient to yield a secure channel when appropriately composed with a secure symmetric encryption algorithm and MAC. As far as we know, this result was never adapted to the password-based case.

Next step in the same direction was made by Brzuska et al. in [26, 115, 24]. They presented a more general framework which allowed showing that BR-style secure key exchange protocols are composable with a wide class of symmetric key protocols under the condition that a public session matching algorithm for the key exchange protocol

[3]See also [32] for relevant work coming from formal methods community.

exists. In subsequent work [25], the authors have shown that even a weaker notion for key exchange protocols than BR-security would still be enough for composition, and apply this to the TLS handshake. As far as we aware, no similar study has been conducted in the password-based setting. With this work, we aim to beginning filling this gap, by first adapting the results of [26].

Composition of simulation-based key exchange. It is known that a simulation-based formalism typically offers stronger security guarantees than its game-based counterpart. In that regard, the simulation-based models of Shoup [109] (for ordinary key exchange) and Boyko et al. [20] (for PAKE) claim to have a "built in" composition guarantee, but this only been informally argued. Later, applying the methodology of Universal Composability (UC) for key exchange [31], a second, stronger simulation-based notion – Universally Composable PAKE – was proposed by [29]. These models' very strong composition guarantees are appealing, but unfortunately the models themselves are harder to work with than the simpler, game-based models. Another shortcoming of this approach is its restrictive nature which yields not overly efficient protocols. For a nice discussion on the limitations of UC and simulation-based AKE in general, we point reader to [25] and [30].

4.1.4 Organization

The rest of the chapter is organized as follows. In Section 4.2 we state which model we use in the rest of the chapter and provide further discussion on the public matching algorithm needed for our result to hold. Section 4.3 states the definition of Symmetric Key Protocols (SKP) and presents the security game for SKP according to the definition of [26]. We discuss in Section 4.4 how the composition of PAKE and SKP protocols is achieved. Section 4.5 demonstrates the suitability of the Real-Or-Random secure password key exchange protocol for composition with symmetric key protocols.

4.2 PAKE Model

4.2.1 Model

As already explained in the introduction, for our composition result to hold we need to work with a PAKE model that is stringer than the FtG model from [9], namely the RoR model that we described in Section 3.3. This model guarantees that the

adversary who does not know the correct password cannot distinguish *any* honestly generated session key from a random key drawn from the key space. In contrast, in the FtG model, only the session key that is targeted by the single available **Test** query is indistinguishable from random. It will become obvious later in Section 4.5 why this difference matters, i.e. why a stronger model is necessary.

4.2.2 Forward Secrecy

As introduced in Section 3.3.3, Forward Secrecy (FS) provides the guarantee that past session keys will not be automatically divulged by the compromise of long-term keys, which in case of PAKE are passwords.

In practice, this property is very useful for data transfer protocols such as TLS or for the composed protocols we are interested in. For example, if forward-secure TLS cipher suites had been systematically used, the impact made by the Heartbleed security bug would have been of a lesser degree - private key leakage would not have jeopardized the session keys of past communications.

Note that in our model we use the weakest notion of corruption (that which only releases password information, see [9] and Fig. 3.3) on purpose to make our result stronger by showing that even PAKEs proven secure in the weaker model can be safely composed with symmetric key protocols.

4.2.3 Partner Matching

As in [26], for our composition result to hold, we need the underlying PAKE protocol to satisfy an additional property that we call *partner matching* (see Section 3.3.3). Roughly, this property ensures that when observing many PAKE interactions over a network, it is publicly possible to determine pairs of communicants holding the same session key.

Let us now explain the reason we need this property to hold for RoR secure PAKEs in order to take advantage of our composition result. In our reductionist proof presented in Section 4.5, we will specify an algorithm, the RoR adversary $\mathcal{B}$, which will simulate the appropriate security game for the adversary $\mathcal{A}$ it is using as a subroutine, in the RoR security game (see Figure 4.8). While only observing and forwarding the communication between $\mathcal{A}$ and the RoR challenger $\mathcal{CH}^{RoR}$, $\mathcal{B}$ has to be able to assign the same key to two partner instances for the rest of $\mathcal{A}$'s simulation to be sound. To accomplish this, $\mathcal{B}$ would need to be capable, at any time, to output a list of all partnered instances. However, even if the PAKE protocol is RoR secure,

this ability is not guaranteed. In other words, RoR security does not imply partner matching.

Finally, recall that in Section 3.3.3 we defined an efficient partner matching algorithm M that as a result returns a partnering list $\mathcal{L}_{pnrP}$ containing full partnering information.

Practical implications. It should be noted that partner matching is closely related to *partnering* definition from Section 3.3.2. Often in PAKE research, partnering is basically defined using session identifiers that are locally computed. In practice, most published PAKEs define these identifiers simply by concatenating the PAKE message flows with their identities. Clearly, this is a publicly checkable criteria. Hence, the condition causes no real limitation to our result.

4.3 Symmetric Key Protocols

In [26], the authors introduced the notion of Symmetric Key Protocols (SKP) as an umbrella term that encompasses two-party protocols whose execution relies solely on a shared symmetric secret key (e.g. authenticated encryption protocols). In our work, we focus on the same class of protocols, or, more precisely, on the security of their composition with PAKEs. This study is of practical value, since in real world applications session keys that originate from PAKEs are typically used in SKP protocols. In this Section, following [26], we first formally define what an SKP entails, and then present the model that captures generic security requirements of SKP.

4.3.1 Symmetric Key Protocol

We formally define an SKP protocol as a pair of algorithms $(KGen, \mathrm{S})$, where $KGen$ is a randomized key generation algorithm, and S is an algorithm that defines the execution of the SKP protocol. The $KGen$ algorithm outputs the session keys from the key space K according to some probability distribution $\mathcal{K}$ when given as input a security parameter κ.

4.3.2 Security Model

We denote G^{sym} a game that captures the security of the symmetric key protocol. Loosely speaking, the security game should allow the adversary to initialize a new honest instance that is equipped with a fresh session key unknown to the attacker.

Then, being in the two-party symmetric setting, the adversary should also be able to initialize a new instance and partner it with another already existing one - this models the prior result of two partner instances having established the same session key[4]. Additionally, the attacker may create as many dishonest instances as it desires equipped with a secret key of its choice.

Participants. In two-party symmetric key protocols, each principal W comes from a finite, nonempty set ID_{skp}. Note that we do not make any assumption on the principal's possession of a long-term secret.

Protocol execution. Similarly to the PAKE case, the protocol S is a probabilistic algorithm that specifies the reaction of parties involved in SKP to messages that appear on the network. The security game mechanism and the assumption on the power of the adversary are analogous to those for PAKE. Thus, we assume that an adversary $\mathcal{A}$ has complete control of the network. Naturally, each principal may run multiple executions of S with different partners and thus we allow an unlimited supply of instances to be initialized for each principal. In this model, the adversary $\mathcal{A}$ may make *at least* the following queries:

- **InitH**(U^i): Upon receiving this query, the challenger $\mathcal{CH}^{sym}$ initiates an instance U^i with a new session key from the $KGen$ algorithm.
- **InitP**(U^i, V^j): This query initiates an honest instance U^i and assigns to it the key held by V^j, making them partners. A restriction in this model is that at any time during a protocol execution $\mathcal{A}$ may only make one such query per instance. Note that this query faithfully models the asymmetry at the time of key acceptance, since in key exchange there is always one instance waiting for the last protocol message.
- **InitC** $(U^i, sk\text{S})$: As a result of this query, a dishonest instance U^i is initialized with the session key of the adversary's choice skS.
- **Send**(U^i, M): The message M is sent to instance U^i by the adversary $\mathcal{A}$. As a response, U^i processes M according to S, updates its corresponding internal state, and outputs a reply. In this model a **Send**$(U^i, \cdot)$ query is valid only if the instance U^i has been already initialized with one of the three **Init** queries. A

[4]The number of instances sharing the same key should be at most two.

Send(U^i, **Start**) query causes the instance U^i to output S's first message. As in PAKE, the purpose of the **Send** query is to model active attacks.

In addition, the validity and format of each query is checked upon receipt.

We emphasize that this is the *minimal set* of queries $\mathcal{A}$ has access to. Additional queries may be needed depending on the specification of the service S provides. Also, we shall see that the **Send** query holds a special role in the definition secure composition.

Internal state and initialization. As in the PAKE setting, the internal state for G^{sym}, presented in Fig. 4.1, is maintained by the challenger $\mathcal{CH}^{sym}$.

Internal State: For $U \in ID_{skp}$ and $i \in \mathbb{N}$, the internal state is maintained as follows:

- **Execution State** EST_{skp}
  - $\star$ $status\mathrm{S}_U^i \in \{pending, running, aborted\}$
  - $\star$ $sk\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$; $pid\mathrm{S}_U^i \in ID_{skp} \cup \{\perp\}$
  - $\star$ $info\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$
- **Game State** GST_{skp}
  - $\star$ $s_U^i \in \{\perp, private, known\}$; $pnr\mathrm{S}_U^i \in ID_{skp} \times \mathbb{N} \cup \{\perp\}$
  - $\star$ $add\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$.

Figure 4.1: The internal state of Symmetric Key Protocols

Execution state. The execution state, necessary for the execution of the protocol S for every user instance, includes at the very least a session key $sk\mathrm{S}_U^i$, a partner identifier $pid\mathrm{S}_U^i$, and the variable $status\mathrm{S}_U^i$ that shows the status of an instance. Depending on the concrete scheme, the execution state may also include additional values, stored in $info\mathrm{S}_U^i$.

Game state. As in [26], the game state, which stores information relevant to the security game's administration, is left under-specified, due to the generic nature of the model. Therefore, in GST_{skp} we only include a flag s_U^i that tracks the status of the session key held by U^i, and a variable $pnr\mathrm{S}_U^i$ that stores the identity of the

partner instance. Both values are set to $\perp$. The status of the session key is upon initialization set to *private*, except in two cases: (1) the instance came into being through an **InitC** query, and (2) the instance was partnered with a dishonest instance through the **InitP** query. In these two cases the status is set to *known*. Finally, we include $add\mathrm{S}_U^i$, for any additional values the game state may require.

Initialization. The initialization procedure of SKP is slightly different than one from PAKE, since here the adversary uses dedicated queries in order to fully initialize an instance. As shown in Fig. 4.2, the internal state is updated based on the type of initialization query.

InitialSKP(1^k): For $U \in ID_{skp}$ and $i \in \mathbb{N}$, the initialization procedure is performed as follows:

- **Initialize** EST_{skp}
  - $\star$ $status\mathrm{S}_U^i := pending$
  - $\star$ $sk\mathrm{S}_U^i,\ pid\mathrm{S}_U^i := \perp$
- **Initialize** GST_{skp}
  - $\star$ $info\mathrm{S}_U^i := \perp$
  - $\star$ $s_U^i,\ pnr\mathrm{S}_U^i := \perp$
  - $\star$ $add\mathrm{S}_U^i := \perp$

Then, if the adversary ask one of **Init** queries, $status\mathrm{S}_U^i$ is set to *running* and rest of the internal state is updated as follows:

- if **InitH**(U^i): $sk\mathrm{S}_U^i \leftarrow KGen$; $s_U^i := private$
- if **InitC**$(U^i, sk\mathrm{S})$: $sk\mathrm{S}_U^i := sk\mathrm{S}$; $s_U^i := known$;
- if **InitP**(U^i, V^j): $sk\mathrm{S}_U^i := sk\mathrm{S}_V^j,\ pid\mathrm{S}_U^i := V$; $s_U^i := s_V^j,\ pnr\mathrm{S}_U^i := V^j$.

Figure 4.2: The initialization procedure for SKP

Partnering. We say that instance U^i is a partner instance to V^j and vice-versa if: (1) $status\mathrm{S}_U^i = running$ and $status\mathrm{S}_V^j = running$; (2) $pid\mathrm{S}_U^i = V$ and $pid\mathrm{S}_V^j = U$; and (3) $sk\mathrm{S}_U^i = sk\mathrm{S}_V^j$.

SYM security. The definition of sym security depends on the protocol in use. In general, we say that $\mathcal{A}$ wins and breaks the sym security of S, if he triggers some event that a particular security definition deems bad. We denote this bad event **sym**.

The *sym*-advantage of $\mathcal{A}$ in breaking S is

$$\mathbf{Adv}_{\mathrm{S}}^{sym}(\mathcal{A}) := \mathbb{P}[\mathbf{sym}] - \triangle, \tag{4.5}$$

for some constant $\triangle$ that depends on S. (Typical values include 1/2 and 0 for e.g. encryption and MACs respectively.) Note that in contrast to the PAKE case, here it is natural to define security as requiring that the adversary's advantage be *negligible*.

4.4 Composition of PAKE with SKP

In Sections 3.3.2 and 4.3.2 we defined Password Authenticated Key Exchange (PAKE) and Symmetric Key Protocols (SKP), and presented their security models. In this section, we first show how the composition of the two is realized on the algorithmic level and then we present a way to model its security. In this composition, PAKE is responsible for the authenticated key establishment between two users. Afterward, the composition runs a SKP that may serve different purposes, such as providing authenticated or confidential channels (or both), etc. In a similar fashion as in [26], the composition works as follows.

4.4.1 Composed Protocol

Recall that in Sections 3.2 and 4.3 we defined a password authenticated key exchange protocol and symmetric key protocol as pairs of algorithms $(PWGen, \mathrm{P})$ and $(KGen, \mathrm{S})$, respectively. Given these, we now define a *composed protocol* as a pair $(CGen, \mathrm{C})$. We instantiate the $CGen$ algorithm as $PWGen$, i.e. the long-term keys of the composition are those from the key exchange, which in the PAKE case are passwords. For simplicity, we shall assume that these are independent and uniformly distributed. In contrast to $CGen$, algorithm C is instantiated as the "natural" composition of both P and S. More precisely, C first runs the PAKE protocol P and whenever an instance accepts after running P, a freshly generated session key is passed as input to the protocol S, which is thereupon performed. The algorithm C will choose appropriate algorithms to run based on the status of an instance, which can be checked through the $status\mathrm{P}_U^i$ and $status\mathrm{S}_U^i$ variables. Namely, if the $status\mathrm{P}_U^i$ of an instance is set to *running*, the P algorithm will be executed. On the other hand, in case $status\mathrm{P}_U^i = accepted$, S will be performed.

4.4.2 Security Model for Composition

Now, given a composed protocol $(CGen, \mathrm{C})$, the next step is to define a security model for it. We denote by G^{com} a security game for the composed protocol, for which there exists a challenger $\mathcal{CH}^{com}$ that will keep the appropriate secret information away from an adversary $\mathcal{A}$ while administrating the security experiment, like in Fig. 4.3. As one may expect, the composed game will borrow elements from both G^{RoR} and G^{sym}.

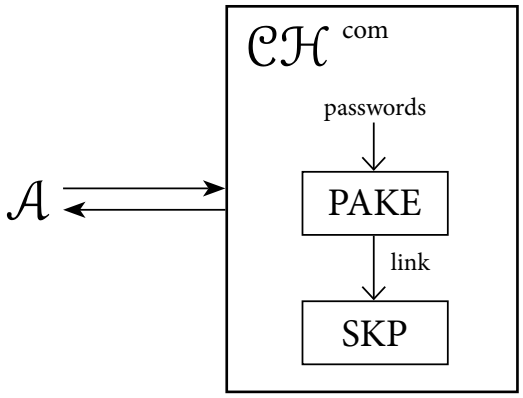

Figure 4.3: Security game for composed protocols

Following [26], we assert that the adversary's ultimate goal against the composition is to break the security of the symmetric key protocol that follows the PAKE. However, since the authentication means in this composed protocol are passwords, it is natural to expect that the best we can hope for in case of an active attacker is that the security definition of the composition is "broken" as soon as a password is guessed.

Participants. The two-party composition of PAKE and SKP inherits PAKE's participant format of disjoint clients and servers. Thus, we will assume that the sets ID_{pake} and ID_{skp} are equal and we denote this single set ID_{com}.

Protocol execution. The protocol C is a probabilistic algorithm that specifies the reaction of principals to network messages. The adversary $\mathcal{A}$ is allowed to interact with multiple distinct executions of both the key exchange and symmetric key protocols. Therefore, we keep the notion of instance intact: An instance of the principal that holds the password - denoted as before U^i - will participate in the execution of the composed protocol. Of course, we also give $\mathcal{A}$ access to certain queries. These are a combination of those in G^{RoR} and G^{sym}.

From G^{RoR}, $\mathcal{A}$ gets access to the **Corrupt** and **Execute** queries, but is no longer allowed to make **Reveal** or **Test** queries. The reason is that in G^{RoR} these latter two queries model session key leakage in higher-level protocols. Since in the composed game the higher-level protocol in question is specified, inheriting these from G^{RoR} is not necessary.

From G^{sym}, $\mathcal{A}$ no longer has access to **InitH**, **InitP**, and **InitC**. This is because intuitively, in the composition, symmetric-key-protocol instance initialization coincides with session-key acceptance. However, $\mathcal{A}$ does have access to any supplementary query available in G^{sym} relevant to S's exact definition. (See the last paragraph of "protocol execution" in 4.3.2.)[5]

Finally, $\mathcal{A}$ still has access to the **Send** query, in order to deliver arbitrary messages to instances. Whether these messages are treated as PAKE messages or symmetric protocol messages depends on the status of the targeted instance: before the instance accepts a session key, it is in "PAKE mode", and after it accepts a session key, it is in "symmetric mode". Upon receipt, a **Send**(U^i, M) query - having the same structure in both phases of composition - is first parsed and checked for validity, and then the message M is processed according to C. Once this is finished and the corresponding internal state of U^i and its partner (if it has one) is updated, the instance outputs a reply that is given to $\mathcal{A}$.

Internal state and initialization. As before, the challenger $\mathcal{CH}^{com}$ will maintain the execution and game state; for G^{com}, these are detailed in Fig. 4.4.

Execution state. The execution state EST_{com} for the composition includes all previously defined variables from that of PAKE and SKP.

Game state. As we mentioned before, we consider the security of the composition to fail if the security of the underlying symmetric key protocol is breached, either *directly* as in G^{sym}, or *via a correct password guess*. Therefore, the game state of the composition GST_{com} includes all G^{sym}-specific variables used to measure G^{sym}-security. Of course, some of the flags from GST_{pake} – such as the partner instance variable, the freshness flag and corruption flags – are also included in GST_{com}.

[5]In particular, if $\mathcal{A}$ has a kind of **Reveal** query in G^{sym}, this is incorporated into the composition game.

Initialization. To initiate the PAKE portion of the internal state we put in use the procedure from Fig. 3.2. As for the SKP portion of the internal state, except $status\mathrm{S}_U^i$ variable that is set as *pending*, most other variables are set to $\perp$ at the start of the game. Only those variables used to measure SKP's security which are initialized at the beginning of G^{sym} are also initialized at the beginning of G^{com}.

Internal State: For $U \in ID_{com}$, $C \in Clients$, $S \in Servers$ and $i \in \mathbb{N}$, the internal state is maintained as follows:

- **Execution State** EST_{com}
  - $\star$ $pw_C \in$ **Pass**; $\quad pw_S := \langle pw_C \rangle_C$
  - $\star$ $status\mathrm{P}_U^i \in \{running, accepted, rejected\}$
  - $\star$ $sid_U^i,\ sk\mathrm{P}_U^i,\ info\mathrm{P}_U^i \in \{0,1\}^* \cup \{\perp\}$; $\quad pid\mathrm{P}_U^i \in ID_{com} \cup \{\perp\}$
  - $\star$ $status\mathrm{S}_U^i \in \{pending, running, aborted\}$
  - $\star$ $sk\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$; $\quad pid\mathrm{S}_U^i \in ID_{com} \cup \{\perp\}$
  - $\star$ $info\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$
- **Game State** GST_{com}
  - $\star$ $pnr\mathrm{P}_U^i \in ID_{com} \times \mathbb{N} \cup \{\perp\}$; $\quad f_U^i \in \{\perp, unfresh, fresh\}$
  - $\star$ $\delta \in \{0,1\}$; $\quad \delta_U^i \in \{honest, corrupted\}$
  - $\star$ $s_U^i \in \{\perp, private, known\}$; $\quad pnr\mathrm{S}_U^i \in ID_{com} \times \mathbb{N} \cup \{\perp\}$
  - $\star$ $add\mathrm{S}_U^i \in \{0,1\}^* \cup \{\perp\}$.

Figure 4.4: The internal state of composed protocols

Bridge between two models. Above, we have shown how the internal state of the composed protocol is defined. Now we need to fuse the two models and protocols together by connecting the queries and the internal states from different phases (PAKE and SKP) with each other.

First, we start by modifying a **Send** query's influence on the internal state. As can be seen in Fig. 4.5, in case the instance U^i accepts as a result of a **Send** query, the corresponding SKP portion of the internal state is initialized by linking it with the internal state from the PAKE phase. Also, it may happen that an instance U^i that

has just accepted with $f_U^i = unfresh$ (due to corruption[6]) has a partner instance V^j with $f_V^j = fresh$. In this case, even though U^i is $unfresh$, we will keep the key s_U^i as *private* (this is incorporated in the partnered case in Fig. 4.5). This is so, since the adversary in this case is not able to influence or learn the session key without breaking the partnering definition. The behavior of the composed game in response to other queries is described in Fig. 4.6.

In addition to the rules from PAKE and SKP for **Send** queries, if an instance U^i changes its $status\mathrm{P}_U^i$ to *accepted* – upon processing its last PAKE message – $\mathcal{CH}^{com}$updates the internal state as follows:

- **Link** EST_{com}
  - $status\mathrm{S}_U^i := running$
  - $sk\mathrm{S}_U^i := sk\mathrm{P}_U^i; \quad pid\mathrm{S}_U^i := pid\mathrm{P}_U^i$
- **Link** GST_{com}
  - $pnr\mathrm{S}_U^i := pnr\mathrm{P}_U^i$
  - s_U^i is initialized as presented below

The status of U^i's session key s_U^i will be initialized (linked) differently depending on which of the following cases occur:

- if there exists a partner instance V^j (according definition from 3.3.2) then:
  - $s_U^i := s_V^j$
- else if $f_U^i = unfresh$ (implies $\delta_U^i = corrupted$) then:
  - $s_U^i := known$
- otherwise,
  - $s_U^i := private$.

Figure 4.5: Linking of the internal state between two phases for the Send query

[6]Notice that in the case an instance has accepted following a **Send** query (see Fig(s). 4.5, 4.7 and 4.10), the flag f_U^i that tracks freshness is solely determined by the corruption flag δ_U^i. This means that if $\delta_U^i = corrupted$, then $f_U^i := unfresh$, and if $\delta_U^i = honest$, then $f_U^i := fresh$. This is so, because it is already assumed for the instance to have $status\mathrm{P}_U^i$ set to *accepted*, and the **Reveal** query is no longer available.

An **Execute**(C^i, S^j) query is simulated by successively running the honest simulations of **Send** queries *up until the acceptance for both instances*. The PAKE internal state is linked as it is for **Send** queries, and s_C^i and s_S^j are both set to *private*.

As a result of the **Corrupt**(U) query, if $U \in Client$ the simulator returns the password pw_C, and otherwise the vector of passwords $pw_S := \langle pw_C \rangle_C$. The corruption flags are affected according to the rules specified in Section 3.3.2.

Figure 4.6: Linking of the Execute query and simulation of the Corrupt query

COM security. As mentioned above, the composition game's security is measured by examining if the protocol S is broken or not. This means that the adversary $\mathcal{A}$ against the composed game G^{com} wins if he satisfies the winning condition for the security game of the underlying symmetric key protocol. Formally, $\mathcal{A}$'s advantage in breaking the composition is defined as

$$\mathbf{Adv}_{\mathrm{C}}^{com}(\mathcal{A}) := \mathbb{P}[\mathbf{sym}] - \triangle, \tag{4.6}$$

where **sym** denotes the same bad event the adversary tries to cause in game G^{sym}. This definition is perfectly valid, since the game state as defined for the composed game incorporates the necessary variables from G^{sym}.

Yet, it is clear that the composed protocol will inherit the limitations built into the use of passwords as key exchange authenticators. Therefore, the best that we can expect is to declare C secure if there exists some positive constant B such that the *com*-advantage of $\mathcal{A}$ in breaking C satisfies

$$\mathbf{Adv}_{\mathrm{C}}^{com}(\mathcal{A}) \ \leq \ \frac{B \cdot n_{se}}{N} + \varepsilon, \tag{4.7}$$

where ε is negligible, and n_{se} is an upper bound on *the number of* ***Send*** *queries* $\mathcal{A}$ *makes to instances where status*P_U^i $=$ *running*. This last point is crucial: Indeed, only those **Send** queries that involved the key exchange phase of the composed protocol should be counted here, since the password's authentication role only plays a part in this phase. Notice that, just like in the case of plain PAKE, this bound implies that **Execute** queries do not significantly contribute to the adversary's advantage.

4.5 Composition Result

This section is devoted to a proof of the following theorem:

Theorem 4.1. *Let* $(PWGen, \mathrm{P})$ *be a password authenticated key exchange protocol outputting keys according to a distribution* $\mathcal{K}$*, that is secure according to the RoR game*

G^{RoR}, *and for which an efficient partner matching algorithm exists. Let* ($KGen$,S) *be a symmetric key protocol secure according to the game* G^{sym}. *If the keys used in the symmetric key protocol algorithm* S *are distributed according to* $\mathcal{K}$, *then the composed protocol* $(CGen, \mathrm{C})$ *is secure according to* G^{com} *and the advantage of any efficient adversary* $\mathcal{A}$ *against composed protocol satisfies*

$$\boldsymbol{Adv}^{com}_{\mathrm{C}}(\mathcal{A}) \;\leq\; \boldsymbol{Adv}^{RoR}_{\mathrm{P}}(\mathcal{B}) \;+\; \boldsymbol{Adv}^{skp}_{\mathrm{S}}(\mathcal{C}) \tag{4.8}$$

for some efficient adversaries $\mathcal{B}$ *and* $\mathcal{C}$.

Proof. Let us fix a polynomial-time adversary $\mathcal{A}$ attacking the protocol C in the security game G^{com}. Our proof is given as a sequence of three hybrid games to bound the advantage of $\mathcal{A}$. Recall that the $KGen$ algorithm outputs the session keys from the key space K according to some probability distribution $\mathcal{K}$ when given as input a security parameter κ.

To prove Theorem 4.1, we first argue that all the session keys computed by the PAKE can be randomized, since the protocol is assumed to be RoR-secure (with weak forward secrecy)[7]. With this step, we will practically decouple the PAKE and SKP phases of the composed protocol, because the session keys that used in the SKP phase of the composition will, from that point on, be completely independent of those computed in the PAKE phase. Then, in the next step, we will show that the advantage of an adversary against the resulting composed game (with random keys) is upper bounded by the advantage of an adversary against the security game of the underlying symmetric key protocol. Let us now proceed with a detailed proof.

Game $\mathbf{G}^{com}_0$: (The original game.) Let this be the game defined in Sect. 4.4.2 for the composed protocol $(CGen, \mathrm{C}_0)$ described in Sect. 4.4.1.

Recall that in this game the adversary $\mathcal{A}$ may make multiple **Send**, **Execute**, and **Corrupt** queries to the challenger. These queries are simulated as described in Sections 3.3.2 and 4.4.2 and Figures 4.5 and 4.6.

Game $\mathbf{G}^{com}_1$: (Key randomization game.) In G^{com}_1, all the session keys skP derived from the PAKE protocol and later used in the SKP portion of the composed protocol are replaced by random keys drawn according to the output distribution of the symmetric key protocol's key generation algorithm, with partnered instances

[7]Note that we do not actually need authentication property to be satisfied for our result to hold.

getting the same random key).

We first explain how the game G_1^{com} differs from G_0^{com}. First, from now on, we assume that the underlying PAKE of the composed protocol C_1 is forward secure in the RoR model. As the **Reveal** and **Test** queries are not available to $\mathcal{A}$ and the **Corrupt** query does not have influence on the security of session keys (due to forward secrecy), the modification brought to G_1^{com} is only related to the **Send** and **Execute** queries and is shown in Fig. 4.7.

Send query modifications:
While all the other rules for internal states remain the same, an instance U^i whose $status\mathrm{P}_U^i$ changed to *accepted* – upon processing its last PAKE message – sets the session key as follows:

- if there exist the partner instance V^j (according definition from 3.3.2) then:
  - $sk\mathrm{P}_U^i := sk\mathrm{P}_V^j$
- else if $f_U^i = unfresh$ then:
  - as before: the session key is computed according to the protocol P (real key)
- otherwise,
  - $sk\mathrm{P}_U^i \leftarrow \mathcal{K}$.

Execute(C^i, S^j) query modifications:
The client instance C^i and the server instance S^j are assigned the same random key drawn according to the output distribution $\mathcal{K}$ of the symmetric key protocol's key generation algorithm $KGen$.

Figure 4.7: The modification of the Send and Execute queries in G_1^{com}

The following lemma will show that by invoking RoR-security, the two games are indistinguishable to the adversary.

Lemma 4.2. *Let* $(PWGen, \mathrm{P})$ *be a password authenticated key exchange protocol outputting keys according to a distribution* $\mathcal{K}$*, that is secure according to the RoR game* G^{RoR}*, and for which an efficient partner matching algorithm exists. Let* $(KGen, \mathrm{S})$ *be a symmetric key protocol, where the keys used in algorithm* S *are distributed according to* $\mathcal{K}$*. The advantage of any efficient adversary* $\mathcal{A}$ *in distinguishing games* G_0^{com} *and*

G_1^{com} *satisfies*

$$\boldsymbol{Adv}_{\mathrm{C}_0}^{com}(\mathcal{A}) \;\leq\; \boldsymbol{Adv}_{\mathrm{C}_1}^{com}(\mathcal{A}) \;+\; \boldsymbol{Adv}_{\mathrm{P}}^{RoR}(\mathcal{B}) \tag{4.9}$$

for some efficient adversary $\mathcal{B}$*.*

Discussion. Before we prove this result, it is worth explaining its contents a bit more. Fundamentally, it is in the statement and proof of this lemma that using the RoR model is important, and where we differ from what is done in [26].

The games G_0^{com} and G_1^{com} are distinct in that the "honest keys" are all real in G_0^{com} and all random in G_1^{com}. If we were to use the FtG model to get from the first game to the second, the fact that FtG only makes *one* **Test** query available to the adversary implies that we would have to use a *hybrid argument* to *gradually* replace the real keys by random ones. This yields an additional security degradation. However, the RoR model lets us move *directly* from G_0^{com} to G_1^{com} by replacing all of the real keys in one swoop. Thus, the degradation vanishes.

Proof. Given an adversary $\mathcal{A}$ against the original game G_0^{com}, we construct an algorithm $\mathcal{B}$ that attempts to break the RoR-security of PAKE (i.e. guess the hidden bit selected by challenger $\mathcal{CH}^{RoR}$) by running the adversary $\mathcal{A}$ as a subroutine, as presented in Figure 4.8. We know that $\mathcal{A}$ asks at most n_{se} **Send**, n_{ex} **Execute**, and n_{co} **Corrupt** queries to its challenger ($\mathcal{B}$ in this case) (resp.) from the composed game interface[8].

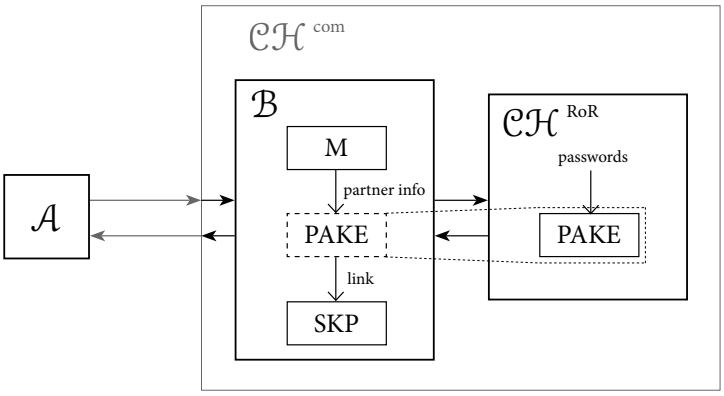

Figure 4.8: Security reduction in $\mathbf{G}_1^{com}$ to prove Lemma 4.2

[8]The dashed box indicates that $\mathcal{B}$ is not actually running PAKE, only tracking the internal state associated with the PAKE phase of the composition.

Main idea behind simulation. Now we need to show that $\mathcal{B}$ can faithfully simulate $\mathcal{A}$'s security game. The main idea is the following: $\mathcal{B}$ directly forwards to the challenger $\mathcal{CH}^{RoR}$ all of $\mathcal{A}$'s queries that are related to the PAKE phase of the composition, while queries from the SKP phase are directly processed by $\mathcal{B}$ using session keys obtained from G^{RoR}. Please note that to this end, we adopt the convention that $\mathcal{B}$ makes an appropriate query to recover a session key as soon as the instance accepts. (Whether this query is a **Test** or **Reveal** query depends on the situation, see further below and Fig(s). 4.10 and 4.11.) When looking in more detail at the simulation, notice that $\mathcal{B}$, in order to safely simulate the SKP phase of the composition, has to initiate and track the internal state of this game (not only session keys) while not having an access to the internal state of G^{RoR}, which will be denoted as $(EST_{pake}, GST_{pake})_{\mathcal{CH}^{RoR}}$. On the bright side, $\mathcal{B}$ has at her disposal all the RoR queries together with the partner matching algorithm. This allows her to keep track of the variables from the internal state of PAKE and store them in $(EST_{pake}, GST_{pake})_{\mathcal{B}}$. These variables are necessary for the initialization of the internal state of the SKP phase $(EST_{skp}, GST_{skp})_{\mathcal{B}}$, which is done through linking of the variables from two phases as defined in Fig(s). 4.5 and 4.6.

Internal state tracking. In Fig. 4.9 is described how $\mathcal{B}$ can obtain information on the internal state of the PAKE protocol that is run by $\mathcal{CH}^{RoR}$. In short, $\mathcal{B}$ does this in three ways: $status\mathrm{P}_U^i$, δ, δ_U^i, and f_U^i are tracked by observing $\mathcal{A}$'s queries and $\mathcal{CH}^{RoR}$'s answers; $sk\mathrm{P}_U^i$, t_U^i, r_U^i, and pw are tracked by making queries such as **Test**, **Reveal** or **Corrupt**; and sid_U^i, $pid\mathrm{P}_U^i$, and $pnr\mathrm{P}_U^i$ are tracked using the public matching algorithm.

Simulation. $\mathcal{B}$ runs the simulation for $\mathcal{A}$ by first, $\mathcal{B}$ initiates her copy of the internal state for both phases of the composition as described in Sect. 4.4.2. Then, the challenger $\mathcal{CH}^{RoR}$ randomly selects a hidden bit b. After this, $\mathcal{A}$ is allowed to start making queries. $\mathcal{B}$ answers to $\mathcal{A}$'s **Send** queries as described in Fig. 4.10. Our assumption that the PAKE protocol is secure in RoR model implies that at least the PAKE partnering definition is satisfied and therefore we can be sure that only two partners from a partnering list $\mathcal{L}_{pnr\mathrm{P}}$ will share the same key. $\mathcal{B}$'s simulation of rest of the queries $\mathcal{A}$ may ask is covered in Fig. 4.11. It is interesting to see that in the case of eavesdropping adversary (no **Send** query), the matching session algorithm is not needed. Notice also that there may exist other possible queries that $\mathcal{A}$ may make – which are added depending on a particular symmetric key protocol. Nevertheless,

$\mathcal{B}$ can obtain information on the internal state of the PAKE protocol as follows:

$status\mathrm{P}_U^i$	The list $\mathcal{L}_{status\mathrm{P}}$ is maintained and $\mathcal{B}$ learns through the inputs of the **Send** queries $\mathcal{A}$ asks if the instance is *running*. Also, the RoR model requires that the challenger $\mathcal{CH}^{RoR}$ provide information to $\mathcal{B}$ if some instance *accepted* or *rejected*.
sid_U^i	Formally, it is not necessary for $\mathcal{B}$ to know this value once she has available partnering information $pnr\mathrm{P}_U^i$ through M. (But, in practice this is easily derived from the message flows.)
$sk\mathrm{P}_U^i$	$\mathcal{B}$ can issue a **Test** or **Reveal** query targeting an instance of $\mathcal{A}$'s interest that has accepted, thus obtaining the session key skP needed for the simulation.
$pid\mathrm{P}_U^i$	This comes from the partner matching algorithm where $pnr\mathrm{P}_U^i$ is parsed and the identity of the partner user is assigned to pidP.
b	This is the bit $\mathcal{B}$ has to try to guess at the end of the game.
t_U^i	$\mathcal{B}$ may make **Test**(U^i) queries, thus he has information about which instance is *tested* or not.
r_U^i	The **Reveal** query is not available to for $\mathcal{A}$. Therefore, $\mathcal{B}$ does not have to track this variable.
$pnr\mathrm{P}_U^i$	Partnering information is acquired through the partner matching algorithm that returns a partnering list $\mathcal{L}_{pnr\mathrm{P}}$.
δ	$\mathcal{B}$ knows when $\mathcal{A}$ makes a **Corrupt** query.
δ_U^i	$\mathcal{B}$ can check if $status\mathrm{P}_U^i = running$ in time when δ changes to 1, and then wait for a **Send**(U^i, M) to update the value assigned to δ_U^i.
f_U^i	Looking at the definition of freshness, condition (1) can be checked since $\mathcal{B}$ tracks $status\mathrm{P}_U^i$, (2) is always satisfied since $\mathcal{A}$ cannot ask **Reveal** queries, and condition (3) information on δ_U^i is available to $\mathcal{B}$.
pw	$\mathcal{B}$ may ask the **Corrupt** query himself and acquire the password or vector of passwords.

In the case of an **Execute**(C^i, S^j) query, C^i and S^j are automatically set to *accepted* and $fresh$, the **Corrupt** query cannot do any harm, and the partnering of the instances is clear from the very nature of the query (thus in this case the partner matching algorithm is not needed).

Figure 4.9: A copy of the internal state for the PAKE portion of the composition

$\mathcal{B}$ can perfectly simulate these queries by using $(EST_{skp}, GST_{skp})_{\mathcal{B}}$.

Upon receipt of a $\textbf{Send}(U^i, M)$ query, $\mathcal{B}$ first checks the value of $status\mathrm{P}_U^i$ in his copy of the internal state. Then, if $status\mathrm{P}_U^i = running$, the $\textbf{Send}(U^i, M)$ query is forwarded to $\mathcal{CH}^{RoR}$, whose response $(status\mathrm{P}_U^{i\prime}, M')$ is given back to $\mathcal{A}$. After that, $\mathcal{B}$ updates the state value to $status\mathrm{P}_U^{i\prime}$ and stores the output message M' in his list of made queries and responses. Furthermore, in case in the received value $status\mathrm{P}_U^{i\prime} = accepted$, $\mathcal{B}$ first calls the partner matching algorithm to obtain a partnering list $\mathcal{L}_{pnr\mathrm{P}}$ and then does the following:

- if there exist the partner instance V^j ($\mathcal{B}$ looks at $\mathcal{L}_{pnr\mathrm{P}}$) then:
  - ⋆ set $sk\mathrm{P}_U^i := skP_V^j$, where skP_V^j was already previously set as a response to a **Reveal** or **Test** query from the cases below.
- else if $f_U^i = unfresh$
  - ⋆ a $\textbf{Reveal}(U^i)$ query is sent to $\mathcal{CH}^{RoR}$ to recover $sk\mathrm{P}_U^i$.
- otherwise,
  - ⋆ a $\textbf{Test}(U^i)$ query is sent to $\mathcal{CH}^{RoR}$ that answers with $sk\mathrm{P}_U^i$. Note that the value of the received $sk\mathrm{P}_U^i$ depends on the hidden bit b.

After acquiring the session key, $\mathcal{B}$ initiates (links) the internal state of SKP for the instance U^i (including s_U^i) according to Fig. 4.5 and continues with the simulation.

Figure 4.10: A Send query simulation by $\mathcal{B}$

Reduction argument. Now, we argue that if the **Test** query that $\mathcal{B}$ issues to $\mathcal{CH}^{RoR}$ returns a real key, then this simulation coincide with the game G_0^{com}. At the same time, if $\mathcal{CH}^{RoR}$ after receiving the **Test** query returns a random key drawn according to $\mathcal{K}$, then $\mathcal{B}$ simulates the G_1^{com} game. W.l.o.g. we will assume that at some point $\mathcal{A}$ will terminate. If $\mathcal{A}$ wins against the composed game, $\mathcal{B}$ will submit $b'' := 1$ to $\mathcal{CH}^{RoR}$, and $b'' := 0$ otherwise. Therefore we have

$$\mathbb{P}^0[b'' = 0] := \mathbf{Adv}_{\mathrm{C}_1}^{com}(\mathcal{A}) \quad \text{and} \quad \mathbb{P}^1[b'' = 1] := \mathbf{Adv}_{\mathrm{C}_0}^{com}(\mathcal{A}). \tag{4.10}$$

By applying Eq. 3.2 we have

Upon receipt of an **Execute**(C^i, S^j) query, $\mathcal{B}$ forwards it to $\mathcal{CH}^{RoR}$, whose response $(status\mathrm{P}_C^{i\,\prime} := accepted,\ status\mathrm{P}_S^{j\,\prime} := accepted,\ M')$ is given back to $\mathcal{A}$. After that, $\mathcal{B}$ issues a **Test** query to $\mathcal{CH}^{RoR}$ targeting either C^i or S^j. Upon receipt of $sk\mathrm{P}$, whose value depends on the hidden bit b, $\mathcal{B}$ may initiate (link) the internal state of SKP for the instances C^i and S^j according to 4.6 (i.e. set s_C^i and s_S^j to *private*) and continue with the simulation.

In case $\mathcal{A}$ makes a **Corrupt**(U) query, $\mathcal{B}$ forwards this query to $\mathcal{CH}^{RoR}$. As an output, the challenger $\mathcal{CH}^{RoR}$ returns the long term secret of the user U, which can be either a password or a vector of passwords. If this is the first **Corrupt** query $\mathcal{A}$ has asked, $\mathcal{B}$ sets $\delta := 1$ in his copy of the internal state. Recall that other corruption flags are changed after corresponding **Send** query is made.

Figure 4.11: Simulation of the Execute and Corrupt queries

$$\mathbf{Adv}_{\mathrm{P}}^{RoR}(\mathcal{B}) := \ \mathbb{P}^1[b'' = 1] - \mathbb{P}^0[b'' = 0] \ := \ \mathbf{Adv}_{\mathrm{C}_0}^{com}(\mathcal{A}) - \mathbf{Adv}_{\mathrm{C}_1}^{com}(\mathcal{A}). \tag{4.11}$$

Thus we can bound the increase in the advantage of the adversary $\mathcal{A}$ from G_0^{com} to G_1^{com} by using a reduction from RoR security of PAKE (with forward secrecy). With this we conclude our proof of Lemma 4.2.

□

Game $\mathbf{G}_2^{com}$: (Reduce G_1^{com} to G^{sym}.) Following lemma will show that the G_1^{com} game in which all session keys coming from PAKE are randomized can be reduced to the security of the symmetric key protocol game G^{sym}.

Lemma 4.3. *Let G_1^{com} be the composed game in which all session keys held by instances running the RoR secure PAKE are computed as per the rules of Fig. 4.7 using a distribution $\mathcal{K}$. Let (KGen, S) be a symmetric key protocol, where the keys that are used in algorithm S are distributed according to $\mathcal{K}$. The advantage of any efficient adversary $\mathcal{A}'$ in winning the G_1^{com} game is bounded by*

$$\boldsymbol{Adv}_{\mathrm{C}_1}^{com}(\mathcal{A}') \ \leq \ \boldsymbol{Adv}_{\mathrm{S}}^{skp}(\mathcal{B}'), \tag{4.12}$$

for some efficient adversary $\mathcal{B}'$.

Proof. Given an adversary $\mathcal{A}'$ against the game G_1^{com}, we construct an algorithm $\mathcal{B}'$ that attempts to break the security of the game G^{sym} by running the adversary $\mathcal{A}'$ as a subroutine. The main idea behind the proof is the following: Since the session keys used in the SKP phase of composed protocol are independent from those coming

from the PAKE phase, $\mathcal{B}'$ can internally simulate the PAKE execution for $\mathcal{A}'$ and then play his game G^{sym} with $\mathcal{A}'$'s SKP phase queries. It is then easy to see that if $\mathcal{A}'$ wins G_1^{com} game, $\mathcal{B}'$ will win his.

Simulation. To prove Lemma 4.3, we need to formally show that $\mathcal{B}'$ can faithfully simulate the security game $\mathcal{A}'$ is attempting to break. To accomplish this, $\mathcal{B}'$ maintains the internal state of the complete composed game G_1^{com}. The simulation of $\mathcal{A}'$'s game goes as follows: First, $\mathcal{B}'$ initiates the PAKE portion of the internal state. Then $\mathcal{B}'$ allows $\mathcal{A}'$ to make query calls. For all queries that $\mathcal{A}'$ makes for the PAKE phase, $\mathcal{B}'$ executes the PAKE algorithm. Once an instance, during the PAKE execution, changes its status to accepted, $\mathcal{B}'$ proceeds according to the rules from Fig. 4.12. As can be seen in the same figure, $\mathcal{B}'$ just forwards $\mathcal{A}'$'s SKP phase queries towards $\mathcal{CH}^{sym}$ and returns the response.

Upon receipt of a **Send**(U^i, M) query, $\mathcal{B}'$ first checks the value of $status\mathrm{P}_U^i$ in his copy of the internal state and does the following:

- If $status\mathrm{P}_U^i = accepted$, then $\mathcal{B}'$ forwards the **Send**(U^i, M) query directly to $\mathcal{CH}^{sym}$ and returns the response back to $\mathcal{A}'$.
- Otherwise, $\mathcal{B}'$ executes the P protocol, updates the internal state, and returns the response $(status\mathrm{P}_U^{i\,\prime}, M')$ to $\mathcal{A}$. Furthermore, in case the new status of the instance U^i is set to $status\mathrm{P}_U^{i\,\prime} := accepted$, $\mathcal{B}'$ does the following:
  - if there exists the partner instance V^j, then $\mathcal{B}'$ sends **InitP**(U^i, V^j) query to $\mathcal{CH}^{skp}$.
  - else if $f_U^i = unfresh$, then $\mathcal{B}'$ issues **InitC** (U^i, skP) query to $\mathcal{CH}^{skp}$, where skP is the key from the internal state of $\mathcal{B}'$.
  - otherwise, $\mathcal{B}'$ makes an **InitH**(U^i) query to $\mathcal{CH}^{skp}$.

Note that after acquiring the session key by executing the PAKE, $\mathcal{B}'$ initiates (links) the internal state of SKP for the instance U^i (including s_U^i) in a similar way as in Fig.4.5.

Upon receipt of a **Execute**(C^i, S^j) query, $\mathcal{B}'$ issues first an **InitH**(C^i) query to $\mathcal{CH}^{skp}$, which is then followed with **InitP**(C^i, S^j) call. Thus, the client instance C^i and the server instance S^j are assigned random keys drawn according to the output distribution $\mathcal{K}$ of $KGen$.

Figure 4.12: The modification of the Send and Execute queries in G_2^{com}

Reduction argument. Now, let us compare two games G_1^{com} and G_2^{com} in relation to the distribution of the session keys used in the SKP phase of the composition. This can be done by looking at Fig.(s) 4.7 and 4.12. There exist three cases to cover: In case of honestly generated keys, in both games, keys that are used in the SKP phase are drawn according to the probability distribution $\mathcal{K}$ (either selecting them directly or through **InitH** query). In case of an unfresh (corrupted) instance, the protocol in both games is using real keys, computed according to the specifications of the protocol[9]. Finally, in case of an instance that has accepted and already has a partner instance, the key from the instance in question gets assigned that partner's session key value, which happens in both games. Therefore, we can conclude that session keys are identically distributed in both games and that the simulation is sound.

W.l.o.g. we will assume that at some point $\mathcal{A}$ will terminate. It is then easy to see that if $\mathcal{A}'$ wins G_1^{com} game, $\mathcal{B}'$ will win G^{sym}. Therefore, we have

$$\mathbf{Adv}^{com}_{\mathrm{C}_1}(\mathcal{A}') \;\leq\; \mathbf{Adv}^{skp}_{\mathrm{S}}(\mathcal{B}'). \tag{4.13}$$

With this we conclude our proof of Lemma 4.3. □

Finally, by combining the Lemmas 4.2 and 4.3 we obtain Theorem 4.1. □

An immediate consequence of our theorem is that preceding a secure symmetric key algorithm with an optimally *RoR-secure* PAKE yields an optimally secure composed protocol according to our definition of composition security. Also, note that to obtain this result, it is absolutely critical to avoid any kind of security degradation in Lemma 4.2. This seems to only be possible by working with RoR rather than FtG.

[9] As an input to **InitC**, $\mathcal{B}'$ provides a key that comes from true simulation of the PAKE protocol execution.

Chapter 5

J-PAKE Revisited

5.1 Introduction

Since the seminal work of Bellovin and Meritt [13], many different protocols have been proposed in the literature to accomplish password-authenticated key exchange. Among them, the J-PAKE protocol [54] has been noticed by practitioners and considered for a wider use in the real-world applications, such as Internet of Things (IoT), due to its patent-free nature, straightforward implementation, and a distinctive design.

5.1.1 Problem

The J-PAKE protocol is quite unique because it integrates Non-Interactive Zero-Knowledge proofs of knowledge (NIZKs in the rest of the chapter) - specifically, Schnorr proofs of knowledge [105] - effectively into its design. However, the presence of these proofs is actually one of the main arguments of J-PAKE's detractors: Indeed, they add more exponentiations to a protocol that already contains many. To make things worse, the communication cost of J-PAKE further burdens the power consumption of the device, which can have significant consequence in case of IoT. A question that can be asked therefore is whether variants of J-PAKE using fewer proofs of knowledge can be found, and how they compare in terms of efficiency to the original protocol.

5.1.2 Our Contribution

We answer these questions by exhibiting two new protocols - which we call RO-J-PAKE and CRS-J-PAKE - that are very similar to J-PAKE, but each uses two less zero-knowledge proofs. We explicitly prove the security of RO-J-PAKE, following a

similar strategy to that of Abdalla et al. in their recent analysis of J-PAKE [1], and show how the proof can be adapted to the case of CRS-J-PAKE. We also provide a more refined analysis of these protocols' efficiency relative to J-PAKE's. We do this by explicitly examining costs depending on which groups are used to deploy the protocol. This is especially important for RO-J-PAKE since it requires hashing into the group in question. Indeed, while on paper, this appears to have no importance, in practice it requires some attention. We treat the cases of Elliptic Curve (EC) groups and Subgroups of Finite Fields (SFFs), since all three protocols require the Decisional Diffie-Hellman (DDH) assumption to hold. In more detail, our findings are as follows.

• ***In terms of provable security:*** RO-J-PAKE and CRS-J-PAKE are asymptotically as secure as J-PAKE against the same kind of adversaries, namely, algebraic adversaries. However, RO-J-PAKE enjoys a looser security proof than J-PAKE and CRS-J-PAKE, essentially because of the addition of a random oracle. CRS-J-PAKE has the tightest proof of the three protocols. See the bounds in Theorems 5.1 and 5.3.

• ***In terms of computational and communication efficiency:*** The apparent computational gain in efficiency that RO-J-PAKE and CRS-J-PAKE enjoy due to their having two less zero-knowledge proofs than J-PAKE can be summarized as follows:

- When all three protocols are instantiated with ECs, CRS-J-PAKE and RO-J-PAKE cost a total of about 8 group-sized exponentiations less than J-PAKE, as shown in Table 5.1. CRS-J-PAKE has a slight edge over RO-J-PAKE, because the latter requires hashing into an EC group. However, experimental results (see Section 5.6) using recent research by Brier et al. [22] shows that this edge can be practically ignored.

- When all three protocols are instantiated with SFFs, CRS-J-PAKE takes 8 group-sized exponentiations less than J-PAKE, but RO-J-PAKE suffers from two additional exponentiations of size comparable to that of the base field's prime - which is typically way larger than the actual group - thus making it much less efficient than J-PAKE in practice, see Table 5.2. This is also due to the need to hash into an SFF. Thus, unless an efficient hashing method is devised, this instantiation of RO-J-PAKE may only have theoretical interest.

- Regardless of the group instantiation, both RO-J-PAKE and CRS-J-PAKE are more efficient than J-PAKE in terms of communication, as they both send four less group elements and two less scalars than J-PAKE does.

RO-J-PAKE and CRS-J-PAKE have a few other (dis)advantages related to their deployability, and that are worth mentioning. See Section 5.7 for more details.

5.1.3 Previous Work

The work most relevant to ours is that by Hao and Ryan [54] introducing J-PAKE. The protocol is under consideration to be included in the ISO/IEC 11770-4 standard [66]. Also, it has been deployed in several commercial products and software libraries (e.g. in Firefox sync [45] (later discontinued), OpenSSL [44], PaleMoon [98], and the Thread network protocol [52] (EC version)), mainly because of its simplicity and patent-free nature. Note that in the original paper the authors only informally argue the security of J-PAKE proposal. The formal analysis of its security had remained elusive until the work of Abdalla et al. in [1]. Our work is heavily inspired by theirs.

Some other work has been devoted to making PAKE more practical for deployment. For instance, in [85] MacKenzie has revisited the PAK protocol [84], showing how to optimize the underlying protocols with EC and SFF implementations. The use of short exponents has also been considered, see [86]. Yet another line of research involved determining the lowest communication costs for standard-model-secure, CRS-based PAKE, see [73, 69]. More recently, work by Abdalla et al. [2] has shown that the computation costs (in terms of number of exponentiations) of many of these protocols can be diminished as well.

5.1.4 Organization

The rest of the chapter is organized as follows. In Section 5.2 we present the original J-PAKE design and protocol specifications. In Sections 5.3 and 5.4 we describe our new protocols, while Section 5.5 provides respective proofs of security. A detailed efficiency analysis of proposed protocols, when deployed with EC and SFF, is presented in Section 5.6. Finally, we conclude the chapter in Section 5.7 with the notes that contain a vision of potential deployment issues.

5.2 The J-PAKE Protocol

As described in [54] and analyzed in [1], the original J-PAKE protocol consists of two message rounds and offers implicit authentication.

Notation. Note that in the description of all three protocols below we will use the following notation. For a given security parameter κ, let $\mathbb{G}$ be a finite multiplicative group[1] of prime order q, such that $|q| := \kappa$. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in $[1 \dots q]$, and all operations on these are performed $\mod q$. Let H_0 will 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. This can be done in various ways (e.g. using max or min functions). Further, we will assume that the client and server always check if the received message is well-formed and if the validity of NIZK proof holds under appropriate label. PK will be an algorithm that generates NIZK proofs and VK will an algorithm used to verify them.

Protocol Description. In the first round, each party generates two random group elements and sends them together with corresponding NIZK proofs of the chosen exponents. A client receives X_3 and X_4 values and computes $\alpha := (X_1 X_3 X_4)^{x_2 pw}$, while a server receives X_1 and X_2 values and computes $\beta := (X_1 X_2 X_3)^{x_4 pw}$. In the second round, the client and the server exchange these α and β values, again with corresponding NIZK proofs. In order to compute the shared secret, both parties first cancel the $g^{x_2 x_4 pw}$ factor from the received value, and then exponentiate what is left to either x_2 (client) or x_4 (server). If everything goes according to the protocol's specification, both parties end up with $K := (X_1 X_3)^{x_2 x_4 pw}$. Finally, the secret key is computed by hashing a pair (e, K), where e is the public random value that has been added to J-PAKE protocol in [1] in order to remove unnecessary random oracle assumption and replace it with a computational randomness extractor. Note that the labels in the original proposal from [54] only include identities of the message originators.

[1] As previously mentioned, the group of interest is either a SFF or EC group. Even though EC group operation is in fact addition, for our convenience protocols will be presented multiplicatively.

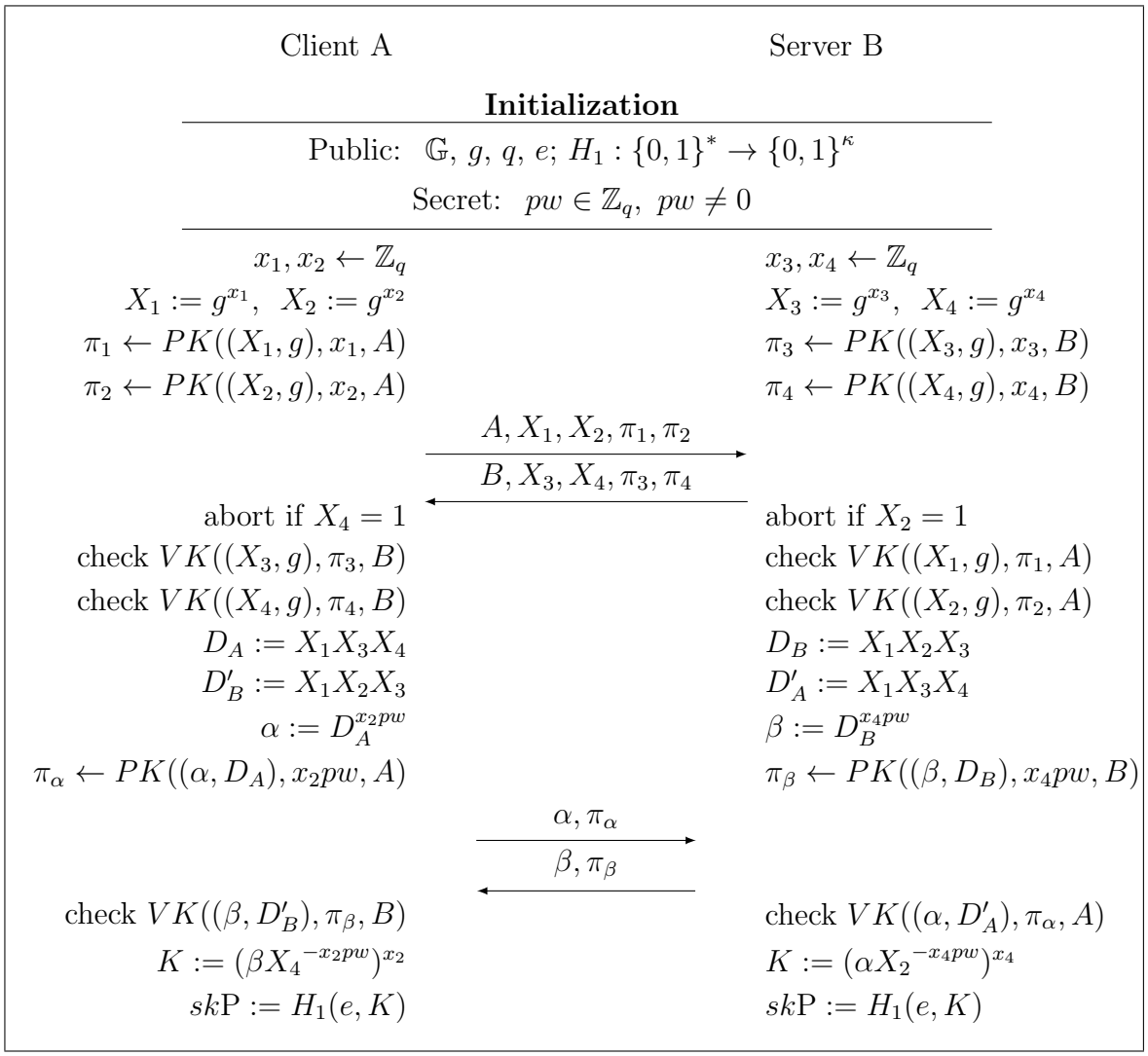

Figure 5.1: The J-PAKE protocol

5.3 The RO-J-PAKE Protocol

When looking at J-PAKE specification on Figure 5.1, one can observe that the exponents x_1 and x_3 are never explicitly used to compute α, β, or K. Parties only need to use the X_1 and X_3 values to generate what can be considered as a random base $T_K := g^{(x_1+x_3)}$ for a Diffie-Hellman (DH) transform[2]. Our idea is to exploit this fact and change the protocol such that the number of NIZK proofs in protocol can be reduced. However, as in the proof of the original J-PAKE (see [1]), we still need to know the discrete logs of X_1 and X_3 for the reduction to work (i.e. in order to simulate the protocol in a sound way). A solution for this is to employ a random oracle taking

[2]To be exact, we should also include pw into the formula for computing the base T_K and thus have $g^{(x_1+x_3)pw}$, but this does not change our claim.

as input fresh messages from each party to provide a random base with exponents known only to the simulator. This idea gives rise to the RO-J-PAKE protocol below.

Protocol Description. A mathematical description of RO-J-PAKE is shown in Fig. 5.2. Next, we rephrase the protocol informally.

After initialization in which public parameters are fixed and a password different from zero is shared between the client and server, the protocol runs in two phases. In the first phase, each party generates one group element and corresponding NIZK proof and sends them – along with its ID – to the other party. In the second phase, upon receiving the first message, both parties compute a common base D as $H_0(f(A, B, X_1, X_2))$. Then, each party computes and sends to other party its commit message that consists of $\alpha := (DX_2)^{x_1 pw}$ and corresponding NIZK proof π_α under label l_A in case of client, and $\beta := (DX_1)^{x_2 pw}$ and π_β under label l_B in case of server. The value of labels are $l_A := (A, B, X_1, X_2)$ and $l_B := (B, A, X_2, X_1)$. Upon receipt of the second message, each party derives a shared secret K, which should be an element of group $\mathbb{G}$, and then a bit-string $sk\mathrm{P}$, which will act as a session key.

Remarks. The purpose of function f is to preserve the symmetry and keep the protocol within two message rounds by making sure that both parties sort values identically and compute the same D. In Sections 5.6 and 5.7, we discuss the instantiation of the hash function H_0, while H_1 with a pair (D, K) as input can be seen as a computational randomness extractor (see Section 2.6.1).

It is worth mentioning that RO-J-PAKE's design prevents the weird-but-benign case of *swapping instances* which happens in the original J-PAKE protocol if the values X_1 and X_2 (or X_3 and X_4 in case of server) are flipped. In that case, the NIZK proof π_β (or π_α resp.) from second message round would still be valid (since the base for the β and α values stay as intended), however, the derived keys would not be the same. A simple solution, proposed in [1], is to expand the NIZK proof labels and add to them all the received values. In RO-J-PAKE, the swapping case does not occur even with the labels left out. However, we strongly advise using the labels in NIZK proofs to ensure that the messages from different rounds are bound together. This additionally makes the proof significantly tighter.

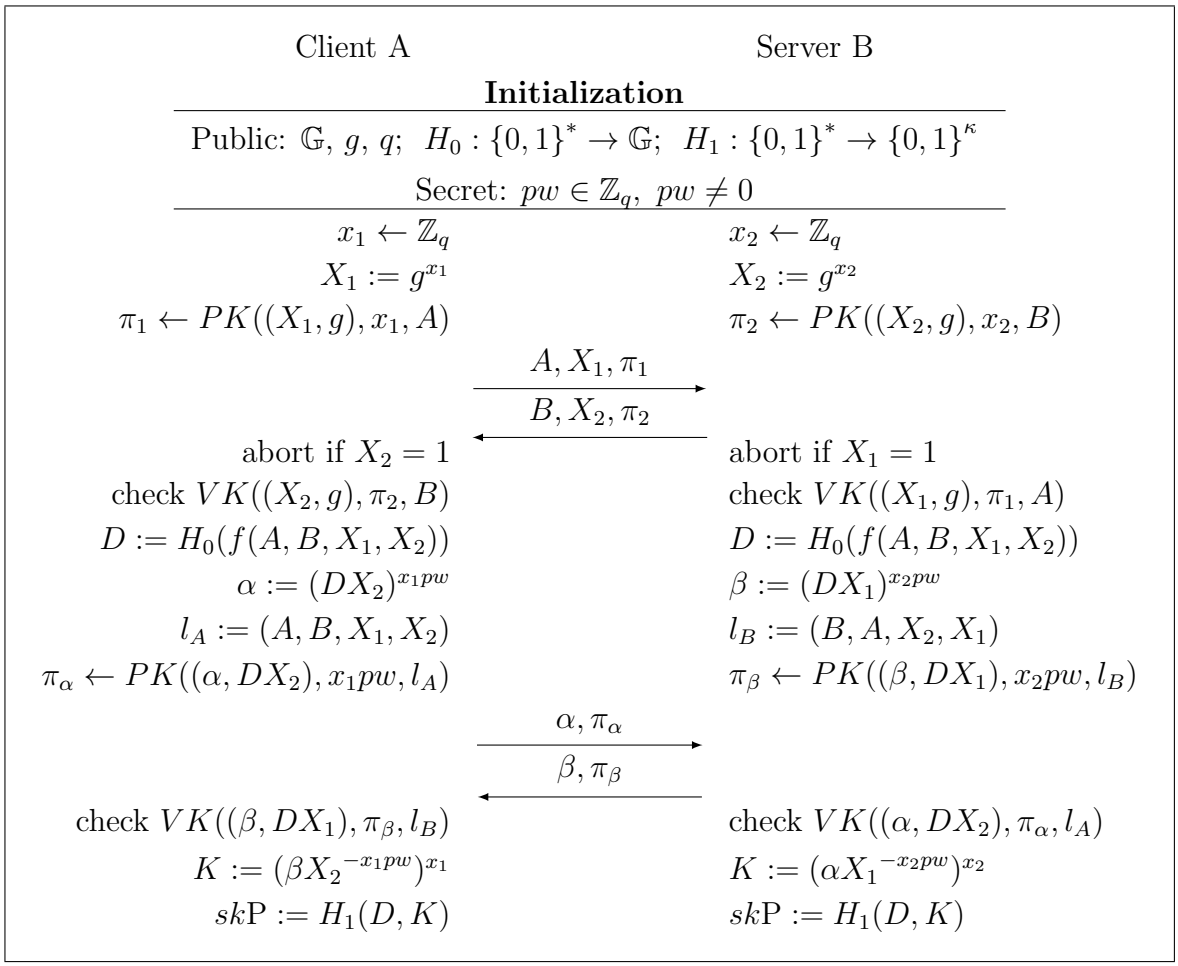

Figure 5.2: The RO-J-PAKE protocol

5.4 The CRS-J-PAKE Protocol

The observation that J-PAKE's X_1X_3 value can be in a sense replaced by a random group element that neither party has control over can be exploited in another direction as well: We can simply add to the protocol's setup a randomly generated value $U \in \mathbb{G}$ that is fixed once and for all, and plays the role of X_1X_3 in J-PAKE and D in RO-J-PAKE for all protocol executions. Hence, we can also consider the CRS-J-PAKE protocol, described fully below. Just like RO-J-PAKE, we eliminate two of the NIZK proofs by design. The name comes from the value U, which is a Common Reference String (CRS). In particular, it carries with it an underlying secret - i.e. the discrete log u of U to the base g - which must be unknown to all parties. In the security proof however, the simulator does get access to u, similarly to the way it knows the discrete logs of the outputs of hash values in the case of RO-J-PAKE (by programming the RO in this way).

Protocol Description. CRS-J-PAKE specification is shown in Fig. 5.3. In comparison to RO-J-PAKE, the major difference is the adoption of the common reference string U, which will be securely chosen in the initialization phase and be hard-coded into the protocol implementation. The client composes its label as $l_A := (A, B, X_1, X_2, U)$ while the server's label is $l_B := (B, A, X_2, X_1, U)$.

Client A		Server B
	Initialization	
Public: $\mathbb{G}$, g, q, $U \in \mathbb{G}$, $e \leftarrow \{0,1\}^t$; $H_1 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$		
Secret: $pw \in \mathbb{Z}_q$, $pw \neq 0$		
$x_1 \leftarrow \mathbb{Z}_q$		$x_2 \leftarrow \mathbb{Z}_q$
$X_1 := g^{x_1}$		$X_2 := g^{x_2}$
$\pi_1 \leftarrow PK((X_1, g), x_1, A, U)$		$\pi_2 \leftarrow PK((X_2, g), x_2, B, U)$
	$A, X_1, \pi_1 \rightarrow$	
	$\leftarrow B, X_2, \pi_2$	
abort if $X_2 = 1$		abort if $X_1 = 1$
check $VK((X_2, g), \pi_2, B, U)$		check $VK((X_1, g), \pi_1, A, U)$
$\alpha := (UX_2)^{x_1 pw}$		$\beta := (UX_1)^{x_2 pw}$
$\pi_\alpha \leftarrow PK((\alpha, UX_2), x_1 pw, l_A)$		$\pi_\beta \leftarrow PK((\beta, UX_1), x_2 pw, l_B)$
	$\alpha, \pi_\alpha \rightarrow$	
	$\leftarrow \beta, \pi_\beta$	
check $VK((\beta, UX_1), \pi_\beta, l_B)$		check $VK((\alpha, UX_2), \pi_\alpha, l_A)$
$K := (\beta {X_2}^{-x_1 pw})^{x_1}$		$K := (\alpha {X_1}^{-x_2 pw})^{x_2}$
$sk\mathrm{P} := H_1(e, K)$		$sk\mathrm{P} := H_1(e, K)$

Figure 5.3: The CRS-J-PAKE protocol

Remarks. As in RO-J-PAKE, swapping instance case does not occur by design. Furthermore, since we no longer need to hash into the underlying group, in contrast to RO-J-PAKE, CRS-J-PAKE has no efficiency issues with respect to a hash implementation. However, the need to generate and trust the hard-coded value U poses its own deployment issues (see Section 5.7).

5.5 Proofs of Security

In this section we present the security proofs for RO-J-PAKE and CRS-J-PAKE. Due to their similarity with the J-PAKE protocol, we are able to structure the proofs in the vein of [1] with the goal to simplify proofreading. The proofs that are demonstrated here are slightly simpler than the original J-PAKE proof. This is true even in case the labels l_A and l_B only contain the identity of the originator of the NIZK proofs π_α and π_β (see later Section 5.5.2), as in the original J-PAKE.

Model. We study the security of RO-J-PAKE and CRS-J-PAKE in the RoR model presented in Section 3.3 that originates from [3]. Being the strongest assumption necessary, we will assume the Decisional Square Diffie Hellman (DSDH, see paragraph 2.7) holds over $\mathbb{G}$. Throughout our analysis, we will assume that the NIZK proofs used are SSE-NIZKPoK (see Section 2.6.3). This is crucial, since it will allow the simulator to tell apart correct and incorrect password guesses and simulate all queries made by the adversary. As in J-PAKE, we keep Schnorr proofs of knowledge as the instantiation of SE-NIZK in our protocols. In [1], they are shown to be SSE-NIZKPoK in the algebraic adversary model with random oracles under one additional condition: the hard-linearity property of bases used in proof must be exhibited. Since the security of our protocols rests on the same hardness assumptions as those in [1], the hard-linearity property of bases is preserved. Additionally, for the proofs to go through, it is as well crucial that the discrete logs of D in RO-J-PAKE (from the RO) and of U in CRS-J-PAKE (the CRS) are known to the simulator. Finally, in the original J-PAKE paper [54], the hash function used for key derivation is implicitly modeled as a random oracle. However, it was shown in [1] that a computational randomness extractor for random group elements from [76] is sufficient. We will follow their lead in our work.

Additionally, for easier comparison with the bounds from [1] we reformulate zero-knowledge property and knowledge extraction property from 2.6.3 and consider it as advantage functions $\mathbf{Adv}^{uzk}_{NIZK}$ and $\mathbf{Adv}^{ext}_{NIZK}$, respectively.

5.5.1 The Proof of Security for RO-J-PAKE

To exhibit the security of RO-J-PAKE, we will bound the adversarial advantage in attacking the ake security of the studied protocols by using sequence-of-games approach. Starting from the original attack game $\mathbf{G}_0$ – which is played between a

challenger $\mathcal{CH}^{RoR}$ and an adversary $\mathcal{A}$ – we will make a small change to a corresponding protocol P_0 and thus define the next game. Our goal is to prove that $\mathcal{A}$'s advantage is proportional to that of the "dummy" online guesser by showing that $\mathcal{A}$ has negligible advantage to distinguish between two successive games with the exception of game $\mathbf{G}_4$, where guessing-the-right-password event occurs with non-negligible probability.[3]

Simulation. Going further, the challenger $\mathcal{CH}^{RoR}$ takes the role of a simulator that executes the protocol for $\mathcal{A}$. The protocol execution begins by an initialization phase that is presented in Figure 5.5. Then, the simulator gives to $\mathcal{A}$ all public values generated in the initialization phase. Upon receiving an oracle query from $\mathcal{A}$, $\mathcal{CH}^{RoR}$ will respond by executing the appropriate algorithm as in Fig. 5.4. All state information generated during the execution of protocol will be recorded by the simulator.

Theorem 5.1. *Consider* ***RO-J-PAKE*** *as specified in Fig. 5.2, with a password set of size N. Let $\mathcal{A}$ be an adversary that runs in time at most t, and makes at most n_{se}, n_{ex}, n_{re}, n_{te}, n_{h0} queries of type* ***Send***, ***Execute***, ***Reveal***, ***Test*** *and RO queries to H_0. It holds that*

$$\begin{aligned}\boldsymbol{Adv}^{ake}_{\mathtt{ro-j-pake}}(\mathcal{A}) \leq \frac{n_{se}}{N} \;+\; O\Bigg(&\frac{(n_{se}+n_{ex}+n_{ho})^2}{q} \;+\; \frac{n_{h0}^2}{q} \;+\; \boldsymbol{Adv}^{dsdh}_{g,\mathbb{G}}(t') \\ &+ (n_{ex}+n_{se}^2)\boldsymbol{Adv}^{dtgdh}_{g,\mathbb{G}}(t') \;+\; 2n_{h0}n_{se}\boldsymbol{Adv}^{ddh}_{g,\mathbb{G}}(t') \\ &+ (n_{re}+n_{te})\boldsymbol{Adv}^{comp}_{ext_R}(t') \;+\; \boldsymbol{Adv}^{uzk}_{NIZK}(t') \;+\; \boldsymbol{Adv}^{ext}_{NIZK}(t')\Bigg)\;,\end{aligned}$$

and where $t' = O(t + (n_{se}+n_{ex}+n_{ho})t_{exp})$ with t_{exp} being the time required for an exponentiation in $\mathbb{G}$.

Proof. From now on, the values that are received by an honest party and possibly coming from $\mathcal{A}$ will be denoted as X_1', α', etc. We say that instance is *matching* if $X_1 = X_1'$ and $X_2 = X_2'$. In that case, the client's hash output D_A will be equal to the server's D_B. Also, we say that instances are *fully matching*, if both message rounds are honestly forwarded by $\mathcal{A}$.

Game $\mathbf{G}_0$: (Original protocol) This game is faithful to Fig. 5.2.

[3]This is where the cost is paid for using a small entropy secret for authentication instead of cryptographically strong authenticator.

Game $\mathbf{G}_1$: (Simulation and extraction) As defined in Sect. 3.3.2, we simulate **Send**, **Execute**, **Reveal**, **Corrupt**, and **Test** queries that $\mathcal{A}$ may make, with the difference now that for **Send** queries, the simulator runs an extractor Ext, which takes as input a NIZK proof that is produced by $\mathcal{A}$, and outputs a corresponding witness. If the extraction fails, so does $\mathcal{A}$. Also, all hash queries to H_0 are answered by maintaining a list $\mathcal{L}_{h0}$ (see Fig. 5.4).

> $\boldsymbol{H_0}$: For each hash query $H_0(w)$, if the same query was previously asked, the simulator retrieves the record (w, D, d) from the list $\mathcal{L}_{h0}$ and answers with D. Otherwise, the answer D is chosen according to the following rule:
>
> $\star$ **Rule** $H_0^{(1)}$
> Choose $d \leftarrow \mathbb{Z}_q$. Compute $D := g^d$ and write the record (w, D, d) to $\mathcal{L}_{h0}$.

Figure 5.4: Simulation of the hash function H_0

From now on, we assume that an instance receiving a non-valid NIZK proof aborts. More importantly, the simulator – by running the extractor Ext – can obtain discrete logs x'_1, x'_2, $x'_1 pw'$, and $x'_2 pw'$ (and thus pw') from corresponding NIZK proofs that are generated by $\mathcal{A}$. Note that we assume that the simulator knows the discrete logarithms of the outputs of H_0 queries.

$$\mathbf{Adv}^{ake}_{\mathtt{ro-j-pake}}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_1}(\mathcal{A}) + O\left(\mathbf{Adv}^{uzk}_{NIZK}(t') + \mathbf{Adv}^{ext}_{NIZK}(t')\right) \quad . \tag{5.1}$$

Game $\mathbf{G}_2$: (Force uniqueness and avoid collisions) In this game, collisions on the partial transcript $((A, X_1, \pi_1),(B, X_2, \pi_2))$ and the H_0 random oracle are avoided.

More precisely, if a value X_1 or X_2 is repeated in the protocol execution or has already appeared in the random oracle query made by $\mathcal{A}$, the protocol halts and $\mathcal{A}$ fails. The same happens if the outputs of distinct H_0 random oracle queries coincide. Both events are bounded with the birthday paradox:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_1}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_2}(\mathcal{A}) + O\left(\frac{(n_{se} + n_{ex} + n_{h0})^2}{q}\right) + O\left(\frac{{n_{h0}}^2}{q}\right) \quad . \tag{5.2}$$

Game $\mathbf{G}_3$: (Allow instance linking) Same as $\mathbf{G}_2$.

As we can see in Fig. 5.2, the values A, B, X_1 and X_2 are all included in the labels l_A and l_B. This renders the game $\mathbf{G}_3$ from the J-PAKE proof unnecessary. Moreover, we show in Section 5.5.2 that the security of RO-J-PAKE still holds even if labels that contain only the identity are used.

$$\mathbf{Adv}^{ake}_{\mathrm{P}_2}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_3}(\mathcal{A}) \ . \tag{5.3}$$

Game $\mathbf{G}_4$: (Check password guesses) If before a **Corrupt** query, $\mathcal{A}$ makes a **Send** query to a non-matching instance containing α' or β' that corresponds to a correct password guess, the protocol halts and $\mathcal{A}$ succeeds.

The crucial observation here is that the simulator can check whether the password guess is correct or not. This is so, since the simulator can obtain discrete logs of X'_1, X'_2, α' and β' by running Ext on the corresponding NIZK proofs. The extraction does not work for the value coming from a reduction, which we will call a *simulated value*, otherwise the simulator could break the hardness assumption trivially. To determine if the password guess is correct, the simulator can proceed as follows: 1) if both X'_1 and α' (or X'_2 and β' in the case of server impersonation) come from $\mathcal{A}$, the simulator extracts two discrete logs from the corresponding NIZK proofs (e.g. x'_1 from π_1 and $x'_1 pw'$ from π_α), divides them and checks whether the result is equal to pw_A; or 2) if one of the values that instance receives is a simulated value (X'_1 or α' and X'_2 or β'), the simulator extracts one discrete log of the value coming from $\mathcal{A}$ and combines it with the correct password pw_A to perform a check against the simulated value.

$$\mathbf{Adv}^{ake}_{\mathrm{P}_3}(\mathcal{A}) \leq \mathbf{Adv}^{ake}_{\mathrm{P}_4}(\mathcal{A}) \ . \tag{5.4}$$

Game $\mathbf{G}_5$: (Randomize session keys for wrong password guesses) In case of an false password guess to a non-matching instance, K is set randomly.

Main idea. The proof is split into two parts. In the first, we set K randomly only in the non-matching client instances in case of a wrong guess – we will call those *target* client instances. We construct an algorithm $\mathcal{D}$ that given a tuple $\langle X, Y \rangle$, where $X \leftarrow g^x$ and $Y \in \mathbb{G}$, attempts to break the DSDH assumption by running $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ simulates the protocol for $\mathcal{A}$ by setting K randomly for all target client instances, and computing K normally for all other client instances as

follows.[4]

Reduction. For a given DSDH instance $\langle \mathrm{X}, \mathrm{Y} \rangle$, any client instance A^i chooses $a, b \leftarrow \mathbb{Z}_q^*$ and sets $X_1 = X^a g^b$. When A^i receives a $\mathbf{Send}(A^i, (B, X_2', \pi_2'))$ query, the simulator extracts x_2' from π_2', computes D_A and sets $\alpha = X^{a(d'+x_2')pw_A}\ g^{b(d'+x_2')pw_A}$. After receiving $\mathbf{Send}(A^i, (\beta', \pi_\beta'))$ and extracting a witness from π_β', the client instance A^i computes $K = X^{ad'x_2'pw'}\ g^{bd'x_2'pw'}\ Y^{a^2x_2'(pw'-pw_A)}\ X^{2abx_2'(pw'-pw_A)}\ g^{b^2x_2'(pw'-pw_A)}$. Note that the value pw' is obtained by dividing extracted witnesses from the NIZK proofs $pw' = Ext(\pi_\beta')/Ext(\pi_2')$, while d' comes from the list $\mathcal{L}_{h0}$ (see Fig. 5.4). In case of matching instances or a correct password guess $K = X^{d_A pw_A x_2'}\ g^{bd'x_2'pw'}$, since pw' is equal to pw_A. If Y is a real DSDH challenge, then the value K will be computed as in $\mathbf{G}_4$. On the other hand, if Y is random and an incorrect password guess is made, then K will be random, since $x_2' \neq 0$ (checked in the protocol, see Fig. 5.2) and $a(pw' - pw_A) \neq 0$.

Now we show that the simulation is sound for any instance of B that generates X_2 and receives possibly simulated X_1' or α'. If any of the two received values is not from A^i, the simulator can extract a witness from the NIZK proof and check whether the password is correct or not. Moreover, if both received values are from A^i, the instances are matching, otherwise π_α would not be valid. Thus, there is no need to check the password in this case. Since the reduction for the second part of proof in case of relevant server instances is analogous, we get the following bound:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_4}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_5}(\mathcal{A}) + \mathbf{Adv}^{dsdh}_{g,\mathbb{G}}(t') \ . \tag{5.5}$$

Game $\mathbf{G}_6$: (Randomize session keys for paired instances) In case of a matching instance, set K randomly (and matching instances get the same K).

Main idea. We use DTGDH (see Sec. 2.7) in a hybrid argument. We build an algorithm $\mathcal{D}$ that randomly chooses indexes $i, j \leftarrow \{1, 2, \ldots, n_{se}\}$ and simulates the protocol by computing K randomly for all (lexicographically) previous instances $(A^{i'}, B^{j'})$ that are fully matching, and setting K normally for all later $(A^{i'}, B^{j'})$.

Reduction. For a pair (A^i, B^j), A^i sets $X_1 = X$, B^j sets $X_2 = Y$, while the value Z is embedded as the output of $H_0(A^i, B^j, X, Y)$. If instances A^i and B^j match, A^i

[4]Due to the labels l_A and l_B, the random self-reducibility of the DSDH assumption can be used. This affects the tightness of the proof: we do not need to use **a** hybrid argument, so the factor n_{se} in front of $\mathbf{Adv}^{dsdh}_{g,\mathbb{G}}(\mathcal{D})$ in the Theorem 5.1 does not appear.

sets $\alpha = (DH_g(X,Z)DH_g(X,Y))^{pw_A}$ and B^j sets $\beta = (DH_g(Y,Z)DH_g(X,Y))^{pw_A}$. If they fully match, the shared secret is computed as $K = W^{pw_A}$. If $W = g^{xyz}$ then this simulates computing K normally. If W is random, then K is random since $pw_A \neq 0$.

We now need to check if the simulation is sound for other possible queries to A^i and B^j. If A^i (resp. B^j) receives non-simulated values X_2' and β' (resp. X_1' and α'), it can extract witnesses from the NIZK proofs, check the password guess, and respond accordingly. If A^i (resp. B^j) receives an X_2' (resp. X_1') value from $\mathcal{A}$ and a simulated β (resp. α), π_β (resp. π_α) would not be valid due to the labels l_B (resp. l_A). Conversely, if after the first message flow the instances are matching ($X_1 = X_1'$ and $X_2 = X_2'$), but α' or β' are from $\mathcal{A}$, the password guess will be wrong. In case a password guess is correct and the simulation continues, a **Corrupt** query has been made due to $\mathbf{G}_4$. Since the simulator can extract x_1' or x_2' in case of impersonation, K can be computed as $\mathcal{A}$ expects it to be. Therefore,

$$\mathbf{Adv}^{ake}_{\mathrm{P}_5}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_6}(\mathcal{A}) + (n_{ex} + n_{se}^2)\mathbf{Adv}^{dtgdh}_{g,\mathbb{G}}(t') \ . \tag{5.6}$$

Game $\mathbf{G}_7$: (Randomize α and β) If there was no **Corrupt** query, set α and β randomly in all instances. In the case of a correct password guess to a non-matching instance (after **Corrupt** query), compute K as the other party would.

Main idea. Again the proof is split into two parts. Firstly, using a reduction from DDH, the α values are set randomly. We construct an algorithm $\mathcal{D}$ that randomly chooses $i \leftarrow \{1, 2, \ldots, n_{se}\}$ and $j \leftarrow \{1, 2, \ldots, n_{ho}\}$, and simulates the protocol for $\mathcal{A}$ by setting an exponent $x_1 pw_A$ in α to be random for all client instances prior to A^i and computing α normally for all client instances after A^i as follows.[5]

Reduction. A^i sets $X_1 = X$, and the challenge Y is embedded as the output of the jth $H_0(A, B, X_1, X_2')$ query. After receiving **Send**$(A^i, (B, X_2', \pi_2'))$, the simulator extracts x_2' to compute $\alpha = Z^{pw_A} X^{x_2' pw_A}$. Clearly, if Z is random, so is α, and if $Z = DH_g(X,Y)$, α is computed as in $\mathbf{G}_6$. If $\mathcal{A}$ succeeds (by guessing b or the correct password), $\mathcal{D}$'s guess to the DDH challenger is ${b_1}' = 1$, and 0 otherwise.

Note that upon receiving β which corresponds to a correct password guess (either from $\mathcal{A}$ or from B), the simulator can make other instances of A compute the key as

[5] In addition to factor n_{se}, which appears in the proof of original J-PAKE [1], there is a security degradation of factor n_{ho}, since in this reduction the simulator also needs to guess the 'right" random oracle query.

$\mathcal{A}$ or B would, even if α is random. This is possible since it can extract x_2', $x_2'pw'$ from the proofs, check the password guess, and run A accordingly.

We now show the simulation is sound for any instance of B that generates X_2 and receives a possibly simulated X_1' or α' from A^i: 1) if both values are from A^i, set K randomly (due to $\mathbf{G}_6$); 2) if either value is not from A^i, the simulator can extract witnesses from the proof, checks if the password is correct (and satisfies $\mathbf{G}_4$) or not (and sets K randomly due to $\mathbf{G}_5$); 3) if some α' corresponding to a correct password guess submitted to an instance of B is not from A^i and the execution continues (a **Corrupt** query has been made), the discrete log of X_1 is known and the simulator can compute the shared secret K_B as $\mathcal{A}$ would.
Since the reduction for the second part of proof in case of relevant server instances is analogous, we get the following bound:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_6}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_7}(\mathcal{A}) + 2n_{h0}n_{se}\mathbf{Adv}^{ddh}_{g,\mathbb{G}}(t') \ . \tag{5.7}$$

Game $\mathbf{G}_8$: (Randomize session keys skP) Set skP randomly in all instances in which K is set randomly (the matching instances get the same skP).

Remember that skP is computed as $H_1(D, K)$ and that D is the output of a random oracle. The games are computationally indistinguishable, since K is random and H_1 is a computational randomness extractor.

$$\mathbf{Adv}^{ake}_{\mathrm{P}_7}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_8}(\mathcal{A}) + (n_{re} + n_{te})\mathbf{Adv}^{comp}_{ext_R}(t') \ . \tag{5.8}$$

This concludes the proof.

□

5.5.2 The Proof of Security for RO-J-PAKE with Partial Labels

Here we prove the security of RO-J-PAKE protocol, even in case that labels l_A and l_B only contain an identity of the originator of NIZK proofs π_α and π_β, as in originally proposed J-PAKE. In this case, the major difference occurs in the game $\mathbf{G}_3$ where we need to show that the adversary can not manipulate X_1 and X_2 values such that non-matching client and server instances compute the same bases for α or β. Also, without full labels, we need to use a hybrid argument in $\mathbf{G}_5$ to reduce to DSDH assumption.

Theorem 5.2. *Consider* ***RO-J-PAKE*** *as specified in Fig. 5.2 with labels l_A and l_B only containing the identity A and B respectively. Let $\mathcal{A}$ be an adversary that runs in time at most t, and makes at most n_{se}, n_{ex}, n_{re}, n_{te}, n_{h0} queries of type* ***Send****,* ***Execute****,* ***Reveal****,* ***Test*** *and RO queries to H_0. It holds that*

$$\begin{aligned}\boldsymbol{Adv}^{ake}_{\mathtt{ro-j-pake}}(\mathcal{A}) \leq \frac{n_{se}}{N} \;+\; O\Bigg(& \frac{(n_{se}+n_{ex}+n_{ho})^2}{q} \;+\; \frac{n_{h0}^2}{q} \;+\; n_{se}\boldsymbol{Adv}^{dsdh}_{g,\mathbb{G}}(t') \\ & + (n_{ex}+n_{se}^2)\boldsymbol{Adv}^{dtgdh}_{g,\mathbb{G}}(t') \;+\; 2n_{h0}n_{se}\boldsymbol{Adv}^{ddh}_{g,\mathbb{G}}(t') \\ & + (n_{re}+n_{te})\boldsymbol{Adv}^{comp}_{ext_R}(t') \;+\; \boldsymbol{Adv}^{uzk}_{NIZK}(t') \;+\; \boldsymbol{Adv}^{ext}_{NIZK}(t')\Bigg),\end{aligned}$$

and where $t' = O(t + (n_{se}+n_{ex}+n_{ho})t_{exp})$ with t_{exp} being the time required for an exponentiation in $\mathbb{G}$.

Game $\mathbf{G}_3$: (Disallow same-base attacks) In case of the client instance A^i that generate X_1 and the server instance B^j that generate X_2, the following events are avoided:

1. If $X_2 \neq X_2'$ after $\mathcal{A}$ has made a $\mathbf{Send}(A^i, (B, X_2', \pi_2'))$ query, and A^i and B^j compute the same base for α.

In this case, the values $D_A = H_0(A, B, X_1, X_2')$ and $D_B = H_0(A, B, X_1', X_2)$ are distinct (even if $X_1 = X_1'$), since we avoid collisions on H_0 in $\mathbf{G}_2$. The client will compute a base for α as $T_A^\alpha = D_A X_2'$, while the server will compute the same base as $T_B^\alpha = D_B X_2$. Remember that π_α contains T_A^α, which server must check before verifying the validity of π_α. The probability that $T_A^\alpha = T_B^\alpha$, or more precisely, that the output of a hash function $H_0(A, B, X_1, X_2')$ is equal to the bad value $D_B X_2 / X_2'$ is n_{ho}/q. Notice also that in sub-case when $X_1 = X_1'$, the base for β computed by the client $T_A^\beta = D_A X_1$ can not be equal to the same base $T_B^\beta = D_B X_1$ computed by the server – also due to $\mathbf{G}_2$.

2. If $X_1 \neq X_1'$ after $\mathcal{A}$ has made a $\mathbf{Send}(B^j, (A, X_1', \pi_1'))$ query, and A^i and B^j compute the same base for β.

As in case above, the output of a hash function $H_0(A, B, X_1', X_2)$ is equal to the bad value $D_A X_1 / X_1'$ with the probability of n_{ho}/q. Notice that in sub-case when $X_2 = X_2'$, $t_A^\alpha \neq t_B^\alpha$ since this would mean that $D_A = D_B$ for distinct inputs, which is avoided in $\mathbf{G}_2$.

3. If $X_1 \neq X_1'$ and $X_2 \neq X_2'$ after $\mathcal{A}$ has made a $\mathbf{Send}(B^j, (A, X_1', \pi_1'))$ and a $\mathbf{Send}(A^i, (B, X_2', \pi_2'))$, such that $T_A^\alpha = T_B^\alpha$ or $T_A^\beta = T_B^\beta$.

The third case is practically covered with previous two, and the analysis is analogous. In the case either of the above three events occur, the protocol halts and $\mathcal{A}$ fails.

$$\mathbf{Adv}^{ake}_{\mathrm{P}_2}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_3}(\mathcal{A}) + O\left(\frac{n_{h0}}{q}\right) \quad . \tag{5.9}$$

Game $\mathbf{G}_5$: (Randomize session keys for wrong password guesses) In case of an incorrect password guess to a non-matching instance, K is set randomly.

Main idea. In this game, the reduction is very close to the one in [1]; we use similar hybrid argument, and split the proof in two parts. At first we set K randomly only in the non-matching client instances in case of a wrong password guess – we will call those *target* client instances. We construct an algorithm $\mathcal{D}$ that given a tuple $\langle X, Y\rangle$, where $X \leftarrow g^x$ and $Y \in \mathbb{G}$, attempts to break DSDH assumption (i.e. determine whether Y is random or $Y = g^{x^2}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ chooses a random index $i \leftarrow \{1, 2, \ldots, n_{se}\}$ and simulates the protocol for $\mathcal{A}$ by setting K randomly for all target client instances prior to A^i, and computing K normally for all client instances after A^i as follows.

Reduction. For a given DSDH instance $\langle$X,Y$\rangle$, the client instance A^i sets $X_1 = X$. When A^i receives $\mathbf{Send}(A^i, (B, X_2', \pi_2'))$ query, the simulator extracts x_2' from valid π_2', computes D_A and sets $\alpha = X^{(d_A + x_2')pw_A}$. After receiving $\mathbf{Send}(A^i, (\beta', \pi_\beta'))$ and extracting a witness from π_β', the client instance A^i computes $K = (\beta' g^{-x_2' x pw_A})^x = X^{d' pw' x_2'} Y^{x_2'(pw' - pw_A)}$. Note that the value pw' is obtained by dividing extracted witnesses from the NIZK proofs $pw' = Ext(\pi_\beta')/Ext(\pi_2')$, while d' comes from the list $\mathcal{L}_{h0}$ (see Fig. 5.4). In case of matching instances or a correct password guess $K = X^{d_A pw_A x_2'}$, since pw' is equal to pw_A. If Y is real DSDH challenge, then the value K will be computed as in $\mathbf{G}_4$. On the other hand, if Y is random and an incorrect password guess is made, then K will be random, since $x_2' \neq 0$ (check is performed in protocol, see Fig. 5.2) and $pw' - pw_A \neq 0$.

We now have to show that simulation is sound for any instance of B that generates X_2 and receives possibly simulated X_1' or α'. If any of two received values is not coming from A^i, the simulator can extract witness from corresponding NIZK proof and check whether password guess is correct or not. Moreover, if both received values are coming from A^i and instances are non-matching, π_α would be valid only if non-matching A^i and B^j would compute the same base for α – meaning that $D_A^i X_2' = D_B^j X_2$. However,

this event is discarded in the $\mathbf{G}_3$, and therefore there is no need to check password guess in this case.
Since the reduction for the second part of proof in case of relevant server instances is analogous, we get the following bound:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_4}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_5}(\mathcal{A}) + n_{se}\mathbf{Adv}^{dsdh}_{g,\mathbb{G}}(t') \ . \tag{5.10}$$

5.5.3 The Proof of Security for CRS-J-PAKE

Due to its very high similarity with the proof from Section 5.5.1, we will only show a sketch of the security proof for CRS-J-PAKE. The main idea behind the proof is that instead of knowing the discrete logs of H_0's output, the simulator knows the discrete log of parameter U.

Theorem 5.3. *Consider **CRS-J-PAKE** (see Fig. 5.3) with a password set of size N and fixed public value U. Let $\mathcal{A}$ be an adversary that runs in time at most t, and makes at most n_{se}, n_{ex}, n_{re}, n_{te}, n_{h0} queries of type **Send**, **Execute**, **Reveal**, and **Test**. It holds that*

$$\begin{aligned}\boldsymbol{Adv}^{ake}_{\mathtt{crs-j-pake}}(\mathcal{A}) \leq \frac{n_{se}}{N} \ + \ O\Bigg(& \frac{(n_{se}+n_{ex})^2}{q} \ + \ (n_{ex}+n_{se}^2)\boldsymbol{Adv}^{dtgdh}_{g,\mathbb{G}}(t') \\ & + \ \boldsymbol{Adv}^{dsdh}_{g,\mathbb{G}}(t') \ + \ 2n_{se}\boldsymbol{Adv}^{ddh}_{g,\mathbb{G}}(t') \ + \ (n_{re}+n_{te})\boldsymbol{Adv}^{comp}_{ext_R}(t') \\ & + \ \boldsymbol{Adv}^{uzk}_{NIZK}(t') \ + \ \boldsymbol{Adv}^{ext}_{NIZK}(t')\Bigg),\end{aligned}$$

where $t' = O(t + (n_{se} + n_{ex} + n_{ho})t_{exp})$ with t_{exp} being the time required for an exponentiation in $\mathbb{G}$.

Proof Summary. At first, the value of the public parameter U is fixed in the initialization phase and its discrete log u is discarded. A first distinction with the RO-J-PAKE proof occurs in game $\mathbf{G}_2$, where we do not need to avoid collisions on the function H_0. Then, in game $\mathbf{G}_3$, the initialization phase is changed as shown in Fig. 5.5. This means that the discrete log u of the public parameter U is now saved and can be used by the simulator during the protocol execution. Game $\mathbf{G}_4$ stays the same as in the RO-J-PAKE proof.

After the initial four games, we come to the "reduction" games $\mathbf{G}_5$, $\mathbf{G}_6$, and $\mathbf{G}_7$, where we use similar reductions from DSDH, DTGDH and DDH respectively (as in RO-J-PAKE proof), but with a minor differences. In all three reductions, the value u is used during the simulation to compute α and K values, instead of using d_A as in RO-J-PAKE. Also, in $\mathbf{G}_6$, the Z value from a DTGDH challenge tuple is inserted in place

of U instead of being inserted as the output of H_0. Similarly, the value Y that comes from the DDH challenge tuple in $\mathbf{G}_7$ is inserted in place of U instead of in the output of H_0. This last change removes the security degradation factor n_{ho} which appears in RO-J-PAKE's bound. Finally, the public random string e, which is generated during the initialization phase, allows us to set $sk\mathrm{P}$ randomly in all instances in which K is set randomly and to argue the computational indistinguishability between games $\mathbf{G}_7$ and $\mathbf{G}_8$.

Proof Sketch. We will keep the meaning of fully-matching instances as in the proof of RO-J-PAKE. The value of the parameter U is fixed in the initialization phase and its discrete log u is discarded.

Game $\mathbf{G}_0$: (Original protocol) This game is faithful to Fig. 5.3.

Game $\mathbf{G}_1$: (Simulation and extraction) Same as in Sec. 5.5.1.

Game $\mathbf{G}_2$: (Force uniqueness and avoid collisions) In this game, collisions on the partial transcript $((A, X_1, \pi_1),(B, X_2, \pi_2))$ are avoided.

We do not need to avoid collisions on the hash function H_0 as in RO-J-PAKE and therefore

$$\mathbf{Adv}^{ake}_{\mathrm{P}_1}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_2}(\mathcal{A}) + O\left(\frac{(n_{se} + n_{ex})^2}{q}\right) \ . \tag{5.11}$$

Game $\mathbf{G}_3$: (Keep the discrete log of public parameter) During the initialization phase, the discrete log u of the public parameter U is saved for future use.

At this point, the only change is made in the initialization procedure as in Fig. 5.5. As a result, the simulator can retrieve the record (U, u) from $\mathcal{L}_U$ during the protocol execution, if necessary. Therefore,

$$\mathbf{Adv}^{ake}_{\mathrm{P}_2}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_3}(\mathcal{A}) \ . \tag{5.12}$$

Game $\mathbf{G}_4$: (Check password guesses) If before a **Corrupt** query, $\mathcal{A}$ makes a **Send** query to a non-matching instance containing α' or β' that corresponds to a correct password guess, the protocol halts and $\mathcal{A}$ succeeds.

As in $\mathbf{G}_4$ from the RO-J-PAKE proof (see Sec. 5.5.1), by using the extraction property of SSE-NIZKPoK, the simulator can check whether the password guess is

Initial(1^κ): Let $\mathbb{G}$ be a finite multiplicative group of prime order q, and g be a generator of $\mathbb{G}$. *Clients* and *Servers* sets are well defined. The initialization procedure is performed as follows:

$\star$ **Rule** *Initialization*$^{(3)}$

Choose $u \leftarrow \mathbb{Z}_q^*$. Compute $U := g^u$ and write the record (U, u) to $\mathcal{L}_U$.
Choose $e \leftarrow \{0,1\}^t$.
For each $A \in Clients$: $pw_A \leftarrow$ **Pass**.
For each $B \in Servers$: $pw_{A,B} = pw_A$.

Return $\mathbb{G}, g, q, e, A, B, U$.

Figure 5.5: The initialization procedure

correct or not.

Game $\mathbf{G}_5$: (Randomize session keys for wrong password guesses) In case of an incorrect password guess to a non-matching instance, K is set randomly.

Main idea. In this game, as in RO-J-PAKE proof, we use the random self-reducibility of DSDH to show that the adversary needs to solve DSDH in order to distinguish between the two games. There is only a minor difference with the RO-J-PAKE proof of $\mathbf{G}_5$. Namely, during the simulation, the value u is used instead of d_A when computing the α and K values.

As before, we split the proof in two parts. In the first part of the proof, we set K randomly only in the non-matching client instances in case of a wrong password guess – we will call those *target* client instances. We construct an algorithm $\mathcal{D}$ that given a tuple $\langle X, Y \rangle$, where $X \leftarrow g^x$ and $Y \in \mathbb{G}$, attempts to break the DSDH assumption (i.e. determine whether Y is random or $Y = g^{x^2}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ simulates the protocol for $\mathcal{A}$ by setting K randomly for all target client instances, and computing K normally for all other client instances as follows.

Reduction. For a given DSDH instance $\langle$X,Y$\rangle$, any client instance A chooses $a, b \leftarrow \mathbb{Z}_q$ and sets $X_1 = X^a g^b$. When A^i receives a **Send**(A^i, (B, X_2', π_2')) query, the simulator extracts x_2' from π_2', and sets $\alpha = X^{a(u+x_2')pw_A} g^{b(u+x_2')pw_A}$. After receiving a **Send**(A^i, (β', π_β')) and extracting a witness from π_β', the client instance A^i computes $K = X^{aux_2'pw'} \; g^{bux_2'pw'} \; Y^{a^2x_2'(pw'-pw_A)} \; X^{2abx_2'(pw'-pw_A)} \; g^{b^2x_2'(pw'-pw_A)}$. Note that the value pw' is obtained by dividing the extracted witnesses from the NIZK proofs $pw' =$

$Ext(\pi'_\beta)/Ext(\pi'_2)$, while u comes from the record $\mathcal{L}_U$ (see Fig. 5.5). In case of matching instances or a correct password guess $K = X^{upw_A x'_2} g^{bux'_2 pw'}$, since pw' is equal to pw_A. If Y is a real DSDH challenge, then the value K will be computed as in the previous game. If Y is random and an incorrect password guess is made, then K will be random, since $x'_2 \neq 0$ (check is performed in protocol, see Fig. 5.2) and $pw' - pw_A \neq 0$.

We now have to show that the simulation is sound for any instance of B that generates X_2 and receives a possibly simulated X'_1 or α'. If either value is not from A^i, the simulator can extract a witness from the corresponding NIZK proof and check whether the password guess is correct or not. Moreover, if both received values are coming from A^i, the instances are matching otherwise π_α would not be valid. Thus, there is no need to check a password guess in this case.

Since the reduction for the second part of proof in case of relevant server instances is analogous, we get the following bound:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_4}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_5}(\mathcal{A}) + \mathbf{Adv}^{dsdh}_{g,\mathbb{G}}(t') \ . \tag{5.13}$$

Game $\mathbf{G}_6$: (Randomize session keys for paired instances) In case of a matching instance, set K randomly (with matching instances holding the same random K).

Main idea. In this game, in order to bound the difference in the adversary's advantage between $\mathbf{G}_5$ and $\mathbf{G}_6$, we reduce from DTGDH by using a hybrid argument in the following fashion. First, we construct an algorithm $\mathcal{D}$ that for a given tuple $\langle X, Y, Z, DH_g(X,Y), DH_g(X,Z), DH_g(Y,Z), W\rangle$ attempts to break the DTGDH assumption (i.e. determine whether W is random or $W = g^{xyz}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ chooses two random indexes $i, j \leftarrow \{1, 2, \ldots, n_{se}\}$ and simulates the protocol for $\mathcal{A}$ by computing K randomly for all lexicographically previous instances of (A^i, B^j) that are fully matching, and setting K normally for all lexicographically subsequent instances of (A^i, B^j).

Reduction. The client instance A^i sets $X_1 = X$, the server instance B^j sets $X_2 = Y$, while the value Z is embedded in place of U. In case of matching instances, A^i sets $\alpha = (DH_g(X,Z)DH_g(X,Y))^{pw_A}$ and B^j sets $\beta = (DH_g(Y,Z)DH_g(X,Y))^{pw_A}$. If the instance is fully matching it computes the shared secret as $K = W^{pw_A}$. If $W = g^{xyz}$ then this simulates computing K normally. If W is random, then K is random since $pw_A \neq 0$.

We now need to check if the simulation is sound for other possible queries to A^i and B^j. If A^i (resp. B^j) receives non-simulated values X_2' and β' (resp. X_1' and α'), it can extract witnesses from the corresponding NIZK and check whether the password guess is correct or not, and respond accordingly. If A^i (resp. B^j) receives an X_2' (resp. X_1') value from the adversary and a simulated β (resp. α), the NIZK proof π_β (resp. π_α) would not be valid due to the labels l_B (resp. l_A). Conversely, if after the first message flow the instances are matching ($X_1 = X_1'$ and $X_2 = X_2'$), but α' or β' are from the adversary, the corresponding password guess will be incorrect.

In case a password guess is correct and the simulation continues, a **Corrupt** query has been made due to $\mathbf{G}_4$. Since the simulator can extract x_1' or x_2' in case of an impersonation attempt, the value K can be computed as the adversary expects it to be. Therefore,

$$\mathbf{Adv}^{ake}_{\mathrm{P}_5}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_6}(\mathcal{A}) + (n_{ex} + n_{se}^2)\mathbf{Adv}^{dtgdh}_{g,\mathbb{G}}(t') \ . \tag{5.14}$$

Game $\mathbf{G}_7$: (Randomize α and β) If there was no **Corrupt** query, set α and β randomly in all instances. In the case of a correct password guess to a non-matching instance (after **Corrupt** query), compute K as the other party would have.

Main idea. Again the proof is split in two parts. Firstly, using a reduction from DDH, the α values are set randomly. We construct an algorithm $\mathcal{D}$ that for a given tuple $\langle X, Y, Z\rangle$ from a DDH challenger, attempts to break the DDH assumption (i.e. determine whether Z is random or $Z = g^{xy}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ chooses a random index $i \leftarrow \{1, 2, \ldots, n_{se}\}$, and simulates the protocol for $\mathcal{A}$ by setting an exponent $x_1 pw_A$ (in the computation of α) to be random for all client instances prior to A^i and computing α normally for all client instances after A^i as follows.

Reduction. The client instance A^i sets $X_1 = X$, while the challenge value Y is embedded in place of U. After receiving **Send**$(A^i, (B, X_2', \pi_2'))$, the simulator extracts x_2' to compute $\alpha = Z^{pw_A} X^{x_2' pw_A}$. It is obvious that if Z is random, α is random, too. Also, if $Z = DH_g(X, Y)$, α is computed as in $\mathbf{G}_6$. In case the adversary $\mathcal{A}$ succeeds in the protocol (by guessing the **Test** bit b or the correct password), $\mathcal{D}$'s guess to the DDH challenger is ${b_1}' = 1$, and ${b_1}' = 0$ otherwise. Note that upon receiving β which corresponds to a correct password guess (either from the adversary or from B), the simulator is able to ensure that other instances of A compute the same key as

the adversary or B would, even if α is random. This is possible since the simulator can extract x_2', $x_2'pw'$ values from the received NIZK proofs and check whether the password guess is correct or not, and complete the simulation of A accordingly.

We now have to show that the simulation is sound for any instance of B that generates X_2 and receives possibly simulated X_1' or α' from A^i: 1) if both values are from A^i, set K randomly (due to $\mathbf{G}_6$); 2) if any of two received values is not from A^i, the simulator can extract a witness from the corresponding NIZK proof, check whether the password guess is correct (and satisfy $\mathbf{G}_4$) or not (and set K randomly due to $\mathbf{G}_5$); 3) if an α' that corresponds to a correct password guess submitted to an instance of B is not from A^i and the execution of the protocol continues (a **Corrupt** query has been made), the discrete log of X_1 is known and the simulator can compute the shared secret K_B as A would.

The reduction for the second part of proof in case of relevant server instances is analogous, so we get the following bound:

$$\mathbf{Adv}^{ake}_{\mathrm{P}_6}(\mathcal{A}) = \mathbf{Adv}^{ake}_{\mathrm{P}_7}(\mathcal{A}) + 2n_{se}\mathbf{Adv}^{ddh}_{g,\mathbb{G}}(t') \ . \tag{5.15}$$

Game $\mathbf{G}_8$: (Randomize session keys skP) Set skP randomly in all instances in which K is set randomly (matching instances get the same skP).

Two games are computationally indistinguishable, since public string e and K are random and H_1 is a computational randomness extractor. This concludes the proof. □

5.6 Efficiency Analysis

In theory, for J-PAKE and the two new variants, the modular exponentiations are the predominant factors in the computation. Hence, the computational cost is estimated based on counting the number of such modular exponentiations. Note that it takes one exponentiation to generate a Schnorr NIZK proof and two to verify it [105]. Referring to the protocol specifications in Fig. 5.1, 5.2, and 5.3, we summarize their complexities in Table 5.1.

Table 5.1: The efficiency comparison of J-PAKE, RO-J-PAKE and CRS-J-PAKE

Protocol	Complexity	
	Communication	Computation
J-PAKE	$12 \times \mathbb{G} + 6 \times \mathbb{Z}_q$	28$\|q\|$-bit exp
RO-J-PAKE	$8 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	20$\|q\|$-bit exp + 2 H_0
CRS-J-PAKE	$8 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	20$\|q\|$-bit exp

In practice however, counting the modular exponentiations is insufficient, in particular for RO-J-PAKE. This is because the true speed depends highly on how H_0 - which lands into the protocol's underlying group - is computed. This is known to be less efficient than just hashing into a set of bitstrings and sometimes tricky to implement, especially in PAKEs. For instance, in protocols such as PAK [20], SPEKE [67] and Dragonfly [61, 79], a password is used as an input to a hash function, which means that the whole hashing procedure must be done in a constant time, otherwise side-channel attacks are possible (i.e. timing attack). This is not the case with RO-J-PAKE, since all the inputs to the hash function are public values, but we still need to address the efficiency concern. Thus, we further discuss the computational complexity with respect to two different instantiations. It is important to recall that for the security proofs to be valid, $\mathbb{G}$ must be such that the DDH assumption is believed to hold, see [17] for examples.

- **SFF instantiation.** Here, we assume that $\mathbb{G}$ is deployed as the q-order subgroup of $GF(p)^*$, where $p = rq + 1$ and p and q are both prime. Thus, we have $|r| = |p| - |q|$. Standard techniques implement H_0 by first hashing into $GF(p)^*$, which is truly cheap, and then *exponentiating the result by* r, which depends on $|r|$. In particular Table 5.1, indicates that J-PAKE is more efficient than RO-J-PAKE if and only if $28|q| \leq 20|q| + 2|r|$, i.e. if and only if $4|q| \leq |r|$. In other words, J-PAKE is better than RO-J-PAKE provided that a single $|r|$-bit exponentiation costs more than 4 $|q|$-bit ones. For example, given a 2048-bit modulus and 224-bit exponents, one $|r|$-bit exponentiation costs a bit over 8 $|q|$-bit ones. One sees the ratio getting much worse as the NIST-recommended [14] parameters grow.

  Since Table 5.2 shows that in general, $|r|$-bit exponentiations cost *way more* than that, J-PAKE is definitely the better option when using SFFs[6]. Note that

[6]Table 5.2 contains some NIST-recommended parameters, but even in theory the situation seems

CRS-J-PAKE performs significantly better than J-PAKE in this setting, having 8 group-sized exponentiations less than J-PAKE.

Table 5.2: Cost of an $|r|$-bit exponentiation compared to a $|q|$-bit one

$\|p\|$	$\|q\|$	$\|r\|$	$\|r\|/\|q\|$
1024	160	856	5.35
2048	224	1824	8.14
3072	256	2816	11

- **EC instantiation.** Recall that the size of parameters used in EC setting is significantly smaller than when working with SFF, while offering equivalent levels of security. Intuitively, when looking at the structure of typically used elliptic curve group and existing hashing algorithms [106, 63, 22], one would expect hashing into elliptic curve to be more efficient that exponentiation operation. In order to test this intuition, we carried out an experiment based on a Win7 64-bit operating system, with Intel(R) Core(TM) i7-5600U CPU@2.60GHz and 8.0GB RAM. In our test, we assumed the EC is over prime field $GF(p)$ with $|p| = 256$, and took $\mathbb{G}$ to be an EC group of prime order q with $|q| > 160$. H_0 was implemented using the recently discovered hashing algorithms of Brier et al. [22]. We found that an exponentiation takes on average 0.001383 seconds, while hashing a message into the EC group only takes 0.000086 seconds. (For reference, the source codes are listed in Appendix A.) This shows that, on average, hashing is about 16 times cheaper than exponentiating. Hence, using ECs, both RO-J-PAKE and CRS-J-PAKE are definitely more efficient than J-PAKE.

5.7 Implementation Notes

On one hand, in favor of the new protocols, both are most-likely patent-free, like their big brother. Indeed, the structure of all three is essentially the same, having nothing really to do with that of EKE [13] or SPEKE [67]. For instance, none of the "J-PAKE"s perform any password-keyed encryption (like EKE) nor do they hash the password to get a commonly agreed-upon base (like SPEKE). The password

hopeless. Indeed, from [17] we see that for the DDH to reasonably hold in an SFF, we actually need $10|q| > |p| = |r|$, rendering the $4|q| \leq |r|$ requirement unachievable.

is not even encrypted, as is done in many PAKEs that are standard-model-secure, e.g. [71, 69].

On the other hand, RO-J-PAKE and CRS-J-PAKE also have specific implementation issues to deal with for their security proofs to be of any use.

- **The random oracle model.** All three protocols' theoretical security relies on the random oracle model which is implicitly present in the security of the Schnorr NIZK proofs. However, RO-J-PAKE uses it arguably more than the other two, because of H_0. This does not point to any particular weakness, but care must always be taken when selecting the hash function in practice. It also introduces a additional degradation factor in the security proof.

- **The CRS.** It is important to understand that CRS-J-PAKE's security relies crucially on the CRS U being generated *randomly* and *such that* $log_g(U)$ *remains unknown to attackers*. This should be done in a trustworthy way [51]. For instance, a trusted authority can be asked to generate U by selecting u at random, setting $U = g^u$, and throwing u away, or even selecting a purely random string μ, and checking that μ encodes good U, without needing to "handle" u at all. This is an option for large institutions trying to deploy this protocol internally for employees. Another option would be for a predetermined set of users to jointly compute U, with the drawback that any additional user would have to trust the generated value.

Chapter 6

Security Analysis of Dragonfly

6.1 Introduction

As we argued throughout this thesis, PAKE research is very active. New protocol designs are regularly proposed and analyzed, and PAKE itself has been subject to standardization since at least 2002 [65]. Although it took some time for more engineering-oriented security community to acknowledge the importance of formal analysis when arguing about the security of newly proposed cryptographic protocols, a recent join effort to develop a robust TLS 1.3 version [100] seams as a big step in the right direction.

6.1.1 Problem

In 2008 Harkins proposed Dragonfly [60] specifically tailored for mesh networks. A standalone variant of this protocol has been submitted for standardization in IETF [61]. Additionally, the first round messages of the Dragonfly protocol (implicit variant of Dragonfly) and its full version were incorporated into several different IETF standards:

- IEEE 802.11 standard [64] for authentication between wireless devices includes Dragonfly in the form of an authentication mode called SAE.
- It is also included in an authentication framework called Extensible Authentication Protocol (EAP), which is frequently used in wireless networks and point-to-point connections. More specifically, Dragonfly is underlying authentication mechanism in EAP-PWD [59].

- In RFC6617 [56], Dragonfly is included as a secure pre-shared key (PSK) authentication method for the Internet Key Exchange Protocol (IKE) when shared secret is password.
- An implicit variant of Dragonfly was incorporated into TLS-like protocol called TLS-Dragonfly that has recently been specified on the IETF [58].

Dragonfly's design is similar to that of Jablon's SPEKE [67]. In 2001, MacKenzie has proved **SPEKE** secure in [83] in simulation-based model from Boyko, MacKenzie and Patel [20]. However, even though two protocols are similar, Dragonfly protocol is more complex than SPEKE having an additional scalar in exchange. As consequence, SPEKE proof from [83] does not apply to Dragonfly. Thus, at the time, the security of Dragonfly was not confirmed by any security proof.

6.1.2 Our Contribution

This thesis exhibits a proof of security in the random oracle model [11] for a protocol similar to the version of Dragonfly that was up for standardization [61]. Thus, we can at least assert that the scheme's main flows - a Diffie-Hellman [38] variant with a password-derived base - are sound. We followed [83]'s proof to structure ours. However, unlike in [83], we incorporated forward secrecy into the analysis, and chose to work in the Bellare et al. model [9]. To our knowledge, this is the first time a protocol employing a password-derived Diffie-Hellman base is proven forward-secure and analyzed using [9].

Shortly after the publication of our proof [79], the Dragonfly protocol has been standardized in IETF in form of RFC7664 [57].

6.1.3 Previous Work

In previous chapter, we have already covered most of the relevant literature in PAKE research. As for Dragonfly, the protocol specifications first appeared in [60]. The attention it has received as an IETF proposal has led it to being broken by Clarke and Hao [34], and subsequently fixed. As we mentioned earlier, the design of Dragonfly is very similar to one from SPEKE [67]. This similarity has naturally appeared in the security proof as well: as in [83], Dragonfly's security is based on the Computational Diffie-Hellman (CDH) and Decisional Inverted-Additive Diffie-Hellman (DIDH) assumptions (see Sect. 2.7).

6.1.4 Organization

The rest of the chapter is structured as follows. First, in Section 6.2 we present the description of the original Dragonfly protocol as specified in [61]. Section 6.3 contains a description of the version of Dragonfly that we later analyze in Section 6.4, by exhibiting the security proof. Finally, a detailed efficiency analysis of Dragonfly will be presented in Section 6.5.

6.2 The Dragonfly Protocol

In this section we present that was submitted for standardization in IETF [61] and eventually accepted as RFC7664 [57]. The cryptographic core of Dragonfly is a Diffie-Hellman key exchange similar to the one used in SPEKE [67], where a function of the password is the base for group values. A high-level protocol description of the original protocol is shown in Figure 6.1.

Notation. Let the function Good(E, s) be true iff: (1) $s \in [1 \dots q]$ and (2) $E \in \mathbb{G}$. We assume the existence of an efficient algorithm to perform the latter check; this is important, as it prevents instantiation-specific attacks, like the small subgroup attack in [34]. Let H_0 be a full-domain hash mapping $\{0,1\}^*$ to $\mathbb{G}$. We also define a hash function H_1 from $\{0,1\}^*$ to $\{0,1\}^{3\kappa}$.

Initialization. At initialization, the password is chosen at random from **Pass** and given to the client and server. Then, both parties compute a base or password element, which is function of a shared password and identities of the protocol participants, by running a "hunting-and-pecking" procedure for the corresponding group.

Protocol Description. In nutshell, the protocol runs in two rounds. In the first round, each participant chooses a random exponent r_i and mask m_i, computes their sum $s_i \in \mathbb{Z}_q$ and the group element $E_i := PW^{-m_i}$, deletes the value of the mask from memory, and sends the commit message (ID, E_i, s_i), where $i = 1, 2$. Upon receiving this message, Good(E_i, s_i) is called to check its validity. If check is sound, both participants derive the Diffie-Hellman value σ, which is equal to $PW^{r_1 r_2}$ in case of an honest exchange. This is followed by a computation of a hash value (using the derived σ value), parsed into three κ-bit strings: an authenticator for each participant and the session key skP. Then, in the second round, the authenticators are exchanged. If the

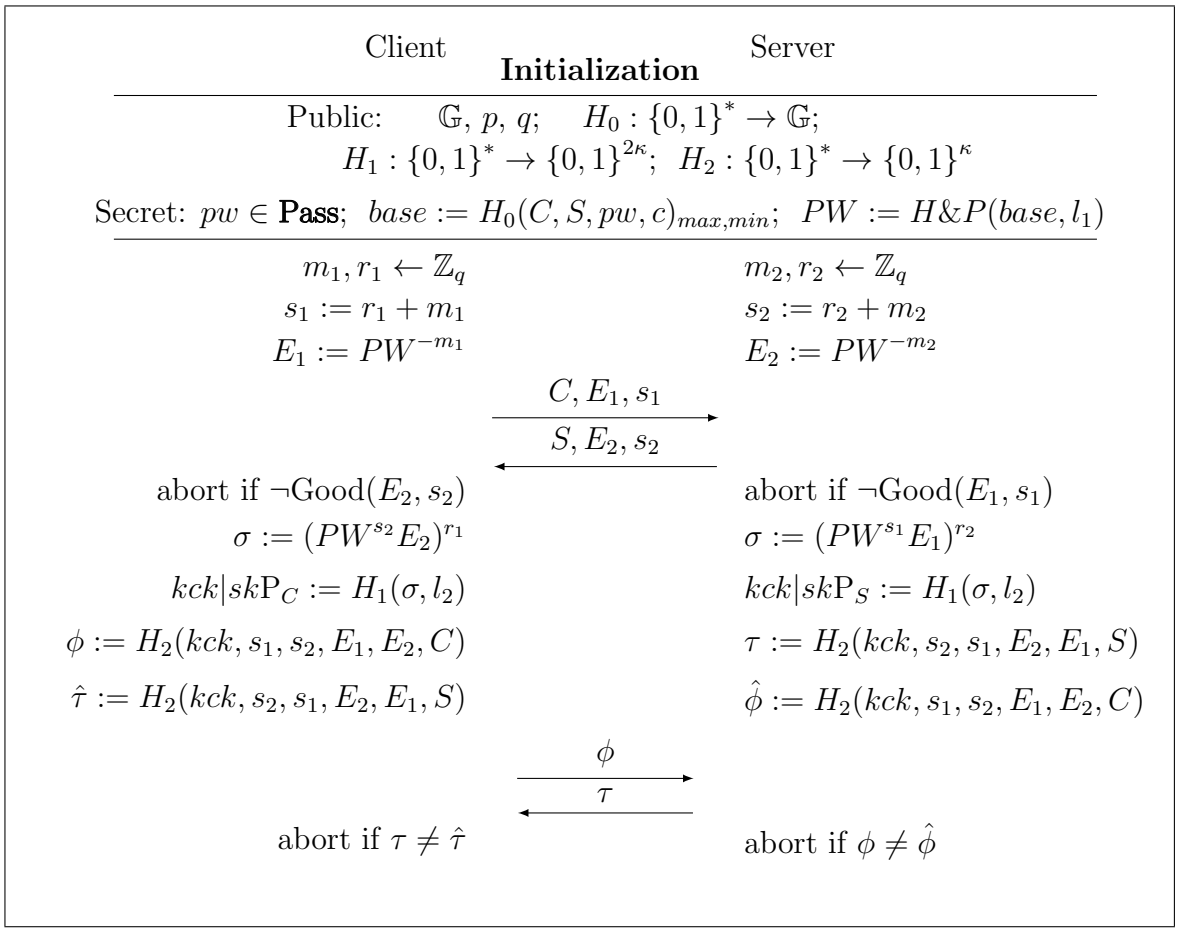

Figure 6.1: The Dragonfly protocol

received authenticator is valid, the participant accepts and terminates the execution, saving session key skP. Otherwise, it aborts, deleting its state.

6.3 The Variant of Dragonfly Protocol

In this section we describe a variant of Dragonfly we will analyze in 6.4. A high-level protocol description is shown in Fig. 6.2.

Initialization. At initialization, the password is chosen at random from **Pass** and given to the client and server. Then, both parties compute a base $PW = H_0(C, S, pw)$ for Diffie-Hellman values, where C and S are ID strings.

Protocol Description. Our the protocol runs in four rounds. In the first round, a client chooses a random exponent r_1 and mask m_1, computes their sum $s_1 \in \mathbb{Z}_q$ and the group element $E_1 := PW^{-m_1}$, and sends the commit message (C, E_1, s_1) to a

server. Upon receiving this message, the server calls $\text{Good}(E_1, s_1)$ to check validity of the first message. Then, if check passes, the server will sends similarly formed message to the client, who will check its validity. At this point, the session IDs sid_C and sid_S are set to $(C, S, s_1, s_2, E_1, E_2)$ for each participant. Now, the client will derive the Diffie-Hellman value $\sigma = PW^{r_1 r_2}$. This is followed by a computation of a hash value (using the derived σ value), parsed into three κ-bit strings: an authenticator for each participant and the session key $sk\text{P}$. Then, the client will send his authenticator. Upon receiving the third round's message, the server will similarly compute the authenticators and the session key, and then check if the received authenticator is valid. If this is the case, the server will accept, send his authenticator to the client and terminate the execution, saving session key $sk\text{P}$. Otherwise, it aborts, deleting its state. Finally, upon receiving the server's authenticator, the client will first accept if check passes, and then terminate the execution, saving session key $sk\text{P}$.

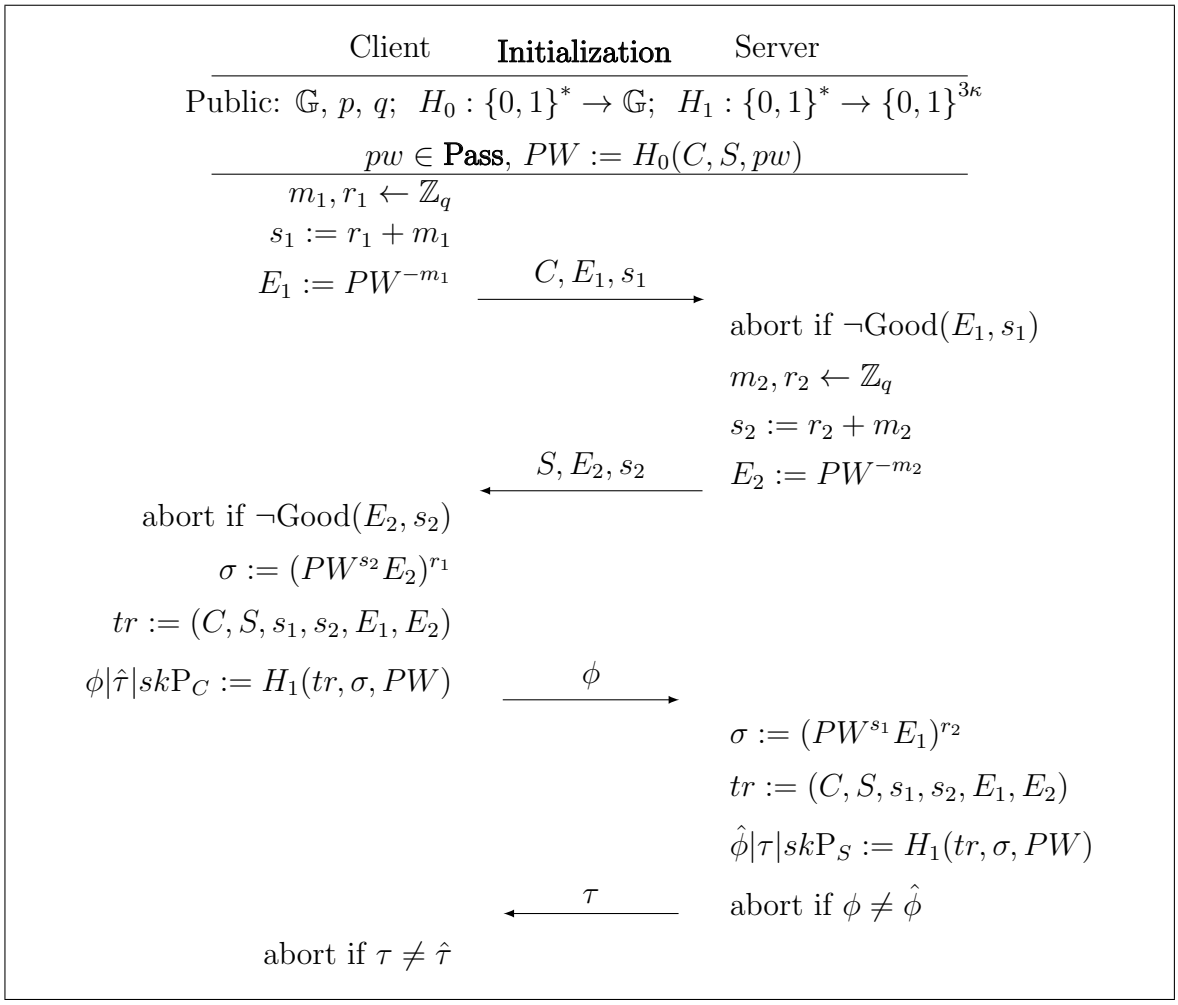

Figure 6.2: The variant of Dragonfly protocol

Remarks. We point out here the main areas where the presented protocol slightly differs from the IETF proposal described in Figure 6.1. First of all, we do not model the "hunting-and-pecking" procedure from [57] explicitly, but this is not a problem here. As pointed out in the proposal, "hunting-and-pecking" is just one way among others to deterministically obtain a base group element from a password. Thus, simply using a random oracle taking as input the participants' identities in addition to the password is appropriate.

In practice, one needs to be very careful when implementing an algorithm that maps the output of hash function to a group. The algorithm must run in constant-time since computations depend on the value of the password and thus can be vulnerable to side-channel attacks. Also, as we could see in 5.6, if $\mathbb{G}$ is deployed as the q-order subgroup of $GF(p)^*$, which is one of the setting recommended in [57], hashing into underlying group does not come cheap in terms of efficiency. Namely, for $p = rq + 1$, the result of a hash function has to be exponentiated with the value of the cofactor r, which, in chosen setting, is roughly 5 times more expensive than an exponentiation with q. Note that in EC setting (also recommended in [57]) this is not an issue.

The procedure we use to compute the confirmation codes and the session keys is not that of the proposal. In particular, our construction makes all of these direct functions of the shared secret, both identities, the main protocol's message flows, and the password element. This is similar to the PAK and PPK protocols [20], for instance, as well as in MacKenzie's analysis of SPEKE [83]. In our view, it is more prudent to follow this pattern, as either removing identities - or replacing them with generic "role" strings - can lead to attacks, e.g. [84] and [1]. Thus, we recommend adding the receiver's identity in the IEFT proposal's computation of the "confirm" message.

Finally, the protocol could have been dropped to three flows, but we chose to keep it four, for two reasons. First, it gives us tighter security reduction, and secondly, despite reducing communication efficiency, four-flow PAKEs - in which the first two flows commit to a shared password and the second two are proofs-of-possession of the session key - are by design secure against many-to-many guessing attacks on the server side, see [78].

6.4 The Security Proof

We now present a proof of security for Dragonfly using the FtG framework defined in 3.3.1. In order to argue the security of our variant of Dragonfly protocol, we will

again employ random oracle model from 2.5.1. We show that Dragonfly distributes semantically secure session keys, provides mutual authentication, and enjoys forward secrecy. We also adopt the convenient notations of [21].

Theorem 6.1. *We consider Dragonfly as described in Section 6.3, with a password set of size N. Let $\mathcal{A}$ be an adversary that runs in time at most t, and makes at most n_{se} **Send** queries, n_{ex} **Execute** queries, and n_{h0} and n_{h1} RO queries to H_0 and H_1, respectively. Then there exist two algorithms $\mathcal{B}$ and $\mathcal{D}$ running in time t' such that $\boldsymbol{Adv}^{ake}_{\texttt{dragonfly}}(\mathcal{A}) \leq T$ and $\boldsymbol{Adv}^{ma}_{\texttt{dragonfly}}(\mathcal{A}) \leq T$ where*

$$T := \frac{6n_{se}}{N} + \frac{4(n_{se}+n_{ex})(2n_{se}+n_{ex}+n_{h1})}{q^2} + \frac{n_{h0}^2+2n_{h1}}{q} + \frac{n_{h1}^2+2n_{se}}{2^\kappa} + 2n_{h1}(1+{n_{se}}^2)Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) + 4n_{h0}^3\left(\boldsymbol{Adv}^{didh}_{g,\mathbb{G}}(\mathcal{D}) + \frac{n_{h1}^3+3n_{se}}{q}\right) \quad (6.1)$$

and where $t' := O(t + (n_{se}+n_{ex}+n_{ro})t_{exp})$ with t_{exp} being a time required for exponentiation in $\mathbb{G}$.

Proof. Our proof is given as a sequence of games $\mathbf{G}_0, ..., \mathbf{G}_4$. Our goal is to prove that Dragonfly resists offline dictionary attacks, i.e. that $\mathcal{A}$'s advantage is proportional to that of the easily detected "dummy" online guesser. We define events, corresponding to $\mathcal{A}$ attacking the protocol in game $\mathbf{G}_m$ and breaking semantic security, and **c2s** and **s2c** authentication, for $m = 0, ..., 4$.

- $\mathbf{S}_m$ occurs if $\mathcal{A}$ returns b' equal to the bit b chosen in the **Test** query.
- $\mathbf{Auth}^{c2s}_m$ occurs if an S^j terminates saving $sk\mathrm{P}_S$ as a state without being partnered with some C^i.
- $\mathbf{Auth}^{s2c}_m$ occurs if a C^i terminates saving $sk\mathrm{P}_C$ as a state without being partnered with some S^j.

Throughout the proof, we call $\mathcal{A}$'s oracle query of the form $H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$ **bad** if $\sigma = DH_{PW}(m, \mu)$ (where $m = PW^{s_1}E_1$ and $\mu = PW^{s_2}E_2$), and $\mathrm{Good}(E_1, s_1)$ and $\mathrm{Good}(E_2, s_2)$ are true. Also, we denote by pw_C the value of the password selected for C and by $PW_{C,S}$ the value of the base derived from it with server S. The number of instances that $\mathcal{A}$ can activate and of hash queries that $\mathcal{A}$ can make are bounded by t. In addition, in case $\mathcal{A}$ does not output b' after time t, b' is chosen randomly. Let us now proceed with a detailed proof.

Game $\mathbf{G}_0$: (Original protocol) This game is our starting point, with Dragonfly defined as in Fig. 6.2. $\mathcal{A}$ may make **Send**, **Execute**, **Reveal**, **Corrupt**, and **Test** queries and these queries are simulated as shown in Fig(s). 6.3, 6.4, and 6.5. From Def. 3.1 we have

$$\mathbf{Adv}^{ake}_{\mathtt{dragonfly}}(\mathcal{A}) := 2\ \Pr[\mathbf{S}_0] - 1\ . \tag{6.2}$$

Send queries made to a client instance C^i are answered as follows:

- A **Send**(C^i, **Start**) query is executed according to the following rule:

  ⋆ **Rule C1**$^{(1)}$
  Choose an ephemeral exponent $r_1 \leftarrow \mathbb{Z}_q$ and a mask $m_1 \leftarrow \mathbb{Z}_q$, compute $s_1 := r_1 + m_1$ and $E_1 := PW^{-m_1}$.

  The client instance C^i then replies to the adversary $\mathcal{A}$ with (C, E_1, s_1) and goes to an expecting state EC_1.

- If the instance C^i is in the expecting state EC_1, a received **Send**(C^i, (S, E_2, s_2)) query is first parsed and Good(E_2, s_2) is called. If the check passes, the instance continues processing the query according to the following rules:

  ⋆ **Rule C2**$^{(1)}$
  Compute $\sigma := (PW^{s_2}E_2)^{r_1}$.

  ⋆ **Rule C3**$^{(1)}$
  Compute $\phi|\hat{\tau}|sk\mathrm{P}_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

  The instance C^i accepts, replies to $\mathcal{A}$ with ϕ, and goes to an expecting state EC_2. Otherwise, it terminates (rejecting), saving no state.

- In case C^i is in the expecting state EC_2, a **Send**(C^i, τ) query is processed according to the following rule:

  ⋆ **Rule C4**$^{(1)}$
  Check if $\tau = \hat{\tau}$. If so, the instance terminates, saving $sk\mathrm{P}_C$ as a state.

  If the equality does not hold, the instance terminates (rejecting), saving no state.

Figure 6.3: Simulation of the Send queries to the client

Send queries made to a server instance S^j are answered as follows:

- A **Send**$(S^j, (C, E_1, s_1))$ query is first parsed and then Good(E_1, s_1) is called. If both values are valid, the instance continues processing the query according to the following rules:

  ⋆ **Rule S1**$^{(1)}$
  Choose an ephemeral exponent $r_2 \leftarrow \mathbb{Z}_q$ and a mask $m_2 \leftarrow \mathbb{Z}_q$, compute $s_2 := r_2 + m_2$ and $E_2 := PW^{-m_2}$.

  The server instance S^j then replies to the adversary $\mathcal{A}$ with (S, E_2, s_2) and goes to an expecting state ES_1. Otherwise, it terminates (rejecting), saving no state.

- If the instance S^j is in the expecting state ES_1, a **Send**(S^j, ϕ) query is executed according to the following rules:

  ⋆ **Rule S2**$^{(1)}$
  Compute $\sigma := (PW^{s_1} E_1)^{r_2}$.

  ⋆ **Rule S3**$^{(1)}$
  $\hat{\phi}|\tau|sk\mathrm{P}_S := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

  ⋆ **Rule S4**$^{(1)}$
  Check if $\phi = \hat{\phi}$. If so, S^j accepts, replies with τ, and terminates while saving $sk\mathrm{P}_S$ as a state.

  If the equality does not hold, the S^j terminates (rejecting), saving no state.

Figure 6.4: Simulation of the Send queries to the server

Game $\mathbf{G}_1$: (Simulation and extraction) This is our first simulation, in which hash queries[1] to H_0, H_1 and H_1' are answered by maintaining lists $\mathcal{L}_{h0}, \mathcal{L}_{h1}$ and $\mathcal{L}'_{h1}$, respectively (see Fig. 6.6). The simulator also maintains a separate list $\mathcal{L}_{\mathcal{A}}$ of all hash queries asked by $\mathcal{A}$. Note that we assume that the simulator knows the discrete logarithms of the outputs of H_0 queries. The simulator also keeps track of all honestly exchanged protocol messages in the list $\mathcal{L}_{\mathcal{P}}$. We say that a client instance C^i and a server instance S^j are paired if $((C, E_1, s_1), (S, E_2, s_2)) \in \mathcal{L}_P$. We can easily see that this simulation is perfectly indistinguishable from the attack in $\mathbf{G}_0$. Thus,

$$\Pr[\mathbf{S}_1] := \Pr[\mathbf{S}_0] \ . \tag{6.3}$$

An **Execute**(C^i, S^j) query is simulated by successively running the honest simulations of **Send** queries. After the completion, the transcript is given to the adversary.
As a result of the **Reveal**(U^i) query, the simulator returns the session key (either $sk\mathrm{P}_C$ or $sk\mathrm{P}_S$) to $\mathcal{A}$, only in case the instance U^i has already computed the key and accepted.
As a result of the **Corrupt**(U) query, if $U \in Client$ the simulator returns the password pw_C, and otherwise the vector of passwords $pw_S = \langle pw_S[C] \rangle_{C \in Client}$.
As a result of the **Test**(U^i) query, the simulator flips a bit b. If $b = 1$, it returns session key $sk\mathrm{P}_U^i$ to $\mathcal{A}$. Otherwise, $\mathcal{A}$ receives a random string drawn from $\{0,1\}^\kappa$.

Figure 6.5: Simulation of the Execute, Reveal, Corrupt, and Test queries

$\boldsymbol{H_0}$: For each hash query $H_0(w)$, if the same query was previously asked, the simulator retrieves the record (w, r, α) from the list $\mathcal{L}_{h0}$ and answers with r. Otherwise, the answer r is chosen according to the following rule: $\star$ **Rule** $H_0^{(1)}$ Choose $\alpha \leftarrow \mathbb{Z}_q$. Compute $r := g^\alpha$ and write the record (w, r, α) to $\mathcal{L}_{h0}$.
$\boldsymbol{H_1}$: For each hash query $H_1(w)$ (resp. $H_1'(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{h1}$ (resp. $\mathcal{L}'_{h1}$) and answers with r. Otherwise, the answer r is chosen according to the following rule: $\star$ **Rule** $H_1^{(1)}$ Choose $r \leftarrow \{0,1\}^{3\kappa}$, write the record (w, r) in the list $\mathcal{L}_{h1}$ (resp. $\mathcal{L}'_{h1}$), and answer with r.

Figure 6.6: Simulation of the hash functions

Game $\mathbf{G_2}$: (Force uniqueness and avoid collisions) In this game, collisions on the outputs of H_0 queries and collisions on the partial transcripts $((C, E_1, s_1), (S, E_2, s_2))$ are avoided. Let list $\mathcal{L}_\mathcal{R}$ keep track of the replies generated by client and server instances as answers to **Send** queries. We abort if a pair $(E_1,\ s_1)$ generated by a client instance is already in the list $\mathcal{L}_\mathcal{R}$ as a result of previous **Send** or **Execute** queries, or in the list $\mathcal{L}_\mathcal{A}$ as an input to an H_1 query. Similarly, we abort in case a pair $(E_2,\ s_2)$ generated by a server instance is already in $\mathcal{L}_\mathcal{R}$ or $\mathcal{L}_\mathcal{A}$.

$\star$ **Rule C1**$^{(2)}$

Choose an ephemeral exponent $r_1 \leftarrow \mathbb{Z}_q$ and a mask $m_1 \leftarrow \mathbb{Z}_q$, compute $s_1 := r_1 + m_1$ and $E_1 := PW^{-m_1}$. If $(E_1,\ s_1) \in \mathcal{L}_\mathcal{R} \cup \mathcal{L}_\mathcal{A}$, abort the game.

[1]The private oracle $H_1' : \{0,1\}^* \to \{0,1\}^{3\kappa}$ will be used in the later, starting from the $\mathbf{G_3}$.

⋆ **Rule S1**$^{(2)}$

Choose an ephemeral exponent $r_2 \leftarrow \mathbb{Z}_q$ and a mask $m_2 \leftarrow \mathbb{Z}_q$, compute $s_2 := r_2 + m_2$ and $E_2 := PW^{-m_2}$. If $(E_2,\, s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

Additionally, we abort in case of collisions on H_0 outputs. This event's probability is bounded by the birthday paradox $n_{h0}^2/2q$.

⋆ **Rule** $H_0^{(2)}$

Choose $\alpha \leftarrow \mathbb{Z}_q$. Compute $r := g^{\alpha}$ and write the record (w, r, α) to $\mathcal{L}_{h0}$. If $(*, r, *) \in \mathcal{L}_{\mathcal{A}}$, abort the game.

The rule modifications in this game ensure the uniqueness of honest instances and that distinct passwords do not map to the same base PW. So we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \frac{2(n_{se} + n_{ex})(2n_{se} + n_{ex} + n_{h1})}{q^2} + \frac{n_{h0}^2}{2q} \quad . \tag{6.4}$$

Game $\mathbf{G_3}$: (Private oracle, avoiding collisions and not guessing authenticators) In this game, we first define event Corrupted that occurs if the adversary makes a Corrupt query while the targeted client and server instance are not paired. From now on, if Corrupted is false, instead of using H_1 to compute session keys and authenticators, the simulator uses a private oracle H_1'. The rules change as follows:

⋆ **Rule C3**$^{(3)}$

If Corrupted is false, compute $\phi|\hat{\tau}|sk\mathrm{P}_C := H_1'(C, S, s_1, s_2, E_1, E_2)$. Otherwise, compute $\phi|\hat{\tau}|sk\mathrm{P}_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

⋆ **Rule S3**$^{(3)}$

If Corrupted is false , compute $\hat{\phi}|\tau|sk\mathrm{P}_S := H_1'(C, S, s_1, s_2, E_1, E_2)$. Otherwise, compute $\hat{\phi}|\tau|sk\mathrm{P}_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

Then, since the shared secret σ and base PW are no longer used in above computations in case the event Corrupted is false, we can further modify the following rules:

⋆ **Rule C2**$^{(3)}$

If Corrupted is false, do nothing. Otherwise, compute $\sigma := (PW^{s_2} E_2)^{r_1}$.

⋆ **Rule S2**$^{(3)}$

If Corrupted is false, do nothing. Otherwise, compute $\sigma := (PW^{s_1} E_1)^{r_2}$.

⋆ **Rule C1**$^{(3)}$

Choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := g^{\psi_1}$. If $(E_1,\, s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort.

⋆ **Rule S1**$^{(3)}$

Choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := g^{\psi_2}$. If $(E_2,\, s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort.

Note that after the above modification the simulator can determine correct and incorrect password guesses and answer perfectly to all queries using the ψ_1, ψ_2, and α values and the lists $\mathcal{L}_{h0}$, $\mathcal{L}_{h1}$, $\mathcal{L}'_{h1}$, $\mathcal{L}_{\mathcal{A}}$, $\mathcal{L}_{\mathcal{P}}$, and $\mathcal{L}_{\mathcal{R}}$. Also, the values s_1, s_2, E_1, E_2 obtained after applying rules $C1^{(3)}$ and $S1^{(3)}$ are identically distributed to those generated in game $\mathbf{G_2}$.

Now that the password-derived base is absent from protocol executions (in case **Corrupted** is false[2]), we can dismiss the event that $\mathcal{A}$ has been lucky in guessing the right $PW_{C,S}$ without making the corresponding H_0 query. Hence we abort the simulation if the adversary $\mathcal{A}$ submits a $H_1(C,\, S,\, *,\, *,\, *,\, *,\, *,\, PW_{C,S})$ query without prior $H_0(C, S, pw_C)$ query. The probability of this event occurring is n_{h1}/q.

⋆ **Rule** $H_1^{(3)}$

If $w = (C, S, *, *, *, *, *, PW_{C,S})$, $((C, S, pw_C), PW_{C,S}) \notin \mathcal{L}_{\mathcal{A}}$, and **Corrupted** is false, abort. Otherwise, choose $r \leftarrow \{0,1\}^{3\kappa}$, write the record (w, r) in the list $\mathcal{L}_{h1}$, and answer with r.

Next, we avoid the cases where the adversary $\mathcal{A}$ may have guessed one of the authenticators (ϕ or τ) without having made an appropriate H_1 query (when **Corrupted** is false). A "lucky guess" occurs if $\mathcal{A}$ submits a **Send**(S^j, ϕ) query with the correct authenticator ϕ to an unpartnered server instance S^j without previously submitting a **bad** $H_1(C,\, S,\, s_1,\, s_2,\, E_1,\, E_2,\, \sigma,\, PW_{C,S})$ query. In this case S^j aborts, even though it should have accepted. Similarly, if $\mathcal{A}$ submits a **Send**(C^i, τ) query with the correct τ to an unpartnered C^i without having submitted a **bad** $H_1(C,\, S,\, s_1,\, s_2,\, E_1,\, E_2,\, \sigma,\, PW_{C,S})$ query, C^i aborts.

⋆ **Rule S4**$^{(3)}$

Check if $\phi = \hat{\phi}$. If so, check if $((C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S}), \phi) \notin \mathcal{L}_{\mathcal{A}}$, where $\mathrm{Good}(E_1, s_1)$ is true, $\sigma = DH_{PW_{C,S}}(m, \mu)$, and **Corrupted** is false. If the latter check is true, the server instance S^j aborts. Otherwise, S^j accepts, replies to the adversary with τ, and terminates while saving $sk\mathrm{P}_S$ as a state.

[2]Notice that in case **Corrupted** is true, a password-derived base is still used in H_1 computations, hence we can not apply the same argument.

⋆ **Rule C4**$^{(3)}$

Check if $\tau = \hat{\tau}$. If so, check if $((C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S}), \tau) \notin \mathcal{L}_{\mathcal{A}}$, where $\mathrm{Good}(E_2, s_2)$ is true, $\sigma = DH_{PW_{C,S}}(m, \mu)$, and **Corrupted** is false. If the latter check is true, the client instance C^i aborts. Otherwise, C^i terminates, saving $sk\mathrm{P}_C$ as a state.

Since the authenticators are computed using a private random oracle H_1' (when **Corrupted** is false), we can argue that the adversary can not do better than a random guess per an authentication attempt via **Send** query. Therefore, the probability of "lucky guessing" is bounded by $n_{se}/2^{\kappa}$.

Without the collisions on the partial transcripts and the "lucky guesses" on the password-derived base and authenticators, one can see that $\mathcal{A}$ has to make the specific combination of H_0 and H_1 hash queries for games $\mathbf{G_2}$ and $\mathbf{G_3}$ to be distinguished. Let $\mathbf{AskH1}_3$ be the event that $\mathcal{A}$ makes the **bad** query $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ PW_{C,S})$ for some transcript $((C, E_1, s_1), (S, E_2, s_2), \phi, \tau)$, where $H_0(C, S, pw_C)$ has been already made. Depending on how the transcript is generated, we distinguish between four disjoint sub-cases $\mathbf{AskH1}_3$:

- $\mathbf{AskH1\text{-}Passive}_3$: $((C, E_1, s_1),\ (S, E_2, s_2),\ \phi,\ \tau)$ comes from an honest execution between C^i and S^j (via an $\mathbf{Execute}(C^i, S^j)$ query);
- $\mathbf{AskH1\text{-}Paired}_3$: $((C, E_1, s_1),\ (S, E_2, s_2))$ comes from an honest execution between C^i and S^j, while $(\phi,\ \tau)$ may come from $\mathcal{A}$;
- $\mathbf{AskH1\text{-}withC}_3$: before any **Corrupt** query, $\mathcal{A}$ interacts with C^i, so (C, E_1, s_1) is generated by C^i, while (S, E_2, s_2) is not from S^j;
- $\mathbf{AskH1\text{-}withS}_3$: before any **Corrupt** query, $\mathcal{A}$ interacts with S^j, so (S, E_2, s_2) is generated by S^j, while (C, E_1, s_1) is not from C^i.

Since session key(s) are computed using the private oracle H_1', the only way $\mathcal{A}$ can break semantic security is via a **Reveal** query to honest instances that generated the same transcript $((C, E_1, s_1), (S, E_2, s_2), \phi, \tau)$, a case we dismissed in $\mathbf{G_2}$. Thus,

$$\Pr[\mathbf{S}_3] := \frac{1}{2} \ , \quad |\Pr[\mathbf{S}_3] - \Pr[\mathbf{S}_2]| \leq \frac{n_{h1}}{q} + \frac{n_{se}}{2^{\kappa}} + \Pr[\mathbf{AskH1}_3] \ . \tag{6.5}$$

Similarly - and as already previously mentioned - the authenticators are computed using H_1' as well, and due to $\mathbf{G_2}$, $\mathcal{A}$ cannot reuse authenticators from other instances. Thus,

$$\Pr[\mathbf{Auth}_3^{c2s}] \leq \frac{n_{se}}{2^{\kappa}} \ , \quad \Pr[\mathbf{Auth}_3^{s2c}] \leq \frac{n_{se}}{2^{\kappa}} \ . \tag{6.6}$$

Game $\mathbf{G_4}$: (Randomize session keys skP)In this game, we estimate the probability of the event $\mathbf{AskH1}_3$ occurring and thus conclude the proof. Notice that the probability of **AskH1** occurring does not change between games $\mathbf{G_3}$ and $\mathbf{G_4}$. We also have that

$$\Pr[\mathbf{S}_4] := \Pr[\mathbf{S}_3] \ , \quad \Pr[\mathbf{Auth}_4^{c2s\ (s2c)}] := \Pr[\mathbf{Auth}_3^{c2s\ (resp.,\ s2c)}] \ . \tag{6.7}$$

Since all the sub-cases of $\mathbf{AskH1}_4$ are disjoint, we will treat them independently:

The following lemma upper bounds the probability of $\mathbf{AskH1\text{-}Passive}_4$:

Lemma 6.2. *For any $\mathcal{A}$ running in time t that asks a bad query $H_1(C,\ S,\ s_1,\ s_2,\ E_1, E_2,\ \sigma,\ PW_{C,S})$ for some transcript $((C, E_1, s_1), (S, E_2, s_2), \phi, \tau)$ that comes from an honest execution between C^i and S^j, there is an algorithm $\mathcal{B}$ running in time $t' := O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the CDH problem:*

$$\Pr[\mathbf{AskH1\text{-}Passive}_4] \leq n_{h1} Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) \ . \tag{6.8}$$

Proof. We construct an algorithm $\mathcal{B}$ that, for given random Diffie-Hellman values $\langle X, Y\rangle$ such that $X \leftarrow g^x$ and $Y \leftarrow g^y$, attempts to break the CDH assumption (i.e. computes Z such that $Z := DH_g(X, Y)$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{B}$ simulates the protocol for $\mathcal{A}$ with the modification of the rules **C1** and **S1** in case an $\mathbf{Execute}(C^i,\ S^j)$ query was made:

- **Rule** $\mathbf{C1}^{(4)}_{exe}$

  Choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := X g^{\psi_1}$. If $(E_1,\ s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

- **Rule** $\mathbf{S1}^{(4)}_{exe}$

  Choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := Y g^{\psi_2}$. If $(E_2,\ s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

After the game ends, for every $H_1(C,\ S,\ s_1,\ s_2,\ E_1, E_2,\ \sigma,\ PW_{C,S})$ query the adversary $\mathcal{A}$ makes, where the values s_1, s_2, E_1 and E_2 were generated by honest client and server instances (after an $\mathbf{Execute}(C^i,\ S^j)$ query), the password-derived base is correct, and the corresponding $H_0(C, S, pw_C)$ query was made,

$$(\sigma Y^{-\frac{\psi_1}{\alpha}} X^{-\frac{\psi_2}{\alpha}} {E_2}^{-s_1} {E_1}^{-s_2} g^{-\frac{\psi_1\psi_2}{\alpha}} g^{-s_1 s_2 \alpha})^{\alpha} \tag{6.9}$$

is added to the list $\mathcal{L}_Z$ of possible values for $Z = DH_g(X, Y)$. Equation 6.9 follows from the fact that a base $PW ::= g^{\alpha}$ is generated in such a way that the discrete

logarithm α is known. Thus, the Diffie-Hellman values X and Y can be represented as $PW^{\frac{x}{\alpha}}$ and $PW^{\frac{y}{\alpha}}$, respectively. So we have:

$$\begin{aligned}\sigma &:= DH_{PW}(E_2 PW^{s_2}, E_1 PW^{s_1}) \\ &= DH_{PW}(PW^{\frac{y+\psi_2}{\alpha}} PW^{s_2}, PW^{\frac{x+\psi_1}{\alpha}} PW^{s_1}) \\ &= Z^{\frac{1}{\alpha}} Y^{\frac{\psi_1}{\alpha}} X^{\frac{\psi_2}{\alpha}} {E_2}^{s_1} {E_1}^{s_2} g^{\frac{\psi_1\psi_2}{\alpha}} g^{s_1 s_2 \alpha} \ .\end{aligned} \tag{6.10}$$

From the adversary's view, the simulation $\mathcal{B}$ runs is indistinguishable from the protocol in the game $\mathbf{G_3}$ up to the point $\textbf{AskH1-Passive}_4$ occurs, and in that case, the correct $DH_g(X, Y)$ value is added to the list $\mathcal{L}_Z$ of size at most n_{h1}. The running time of $\mathcal{B}$ is $t' := O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$. Thus, Lemma 6.2 follows from the fact that the probability of $\mathcal{B}$ breaking CDH assumption is at least $\Pr[\textbf{AskH1-Passive}_4]/n_{h1}$.

□

The next lemma bounds the chance of $\textbf{AskH1-Paired}_4$ occurring:

Lemma 6.3. *For any $\mathcal{A}$ running in time t that asks a bad query $H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S})$ for some partial transcript $((C, E_1, s_1), (S, E_2, s_2)) \in \mathcal{L}_{\mathcal{P}}$ that comes from an honest execution between C^i and S^j, there is an algorithm $\mathcal{B}$ running in time $t' := O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the CDH problem:*

$$\Pr[\textbf{AskH1-Paired}_4] \leq n_{se}^2 n_{h1} Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) \ . \tag{6.11}$$

Proof. The proof is similar to the previous one, except that the simulator needs to guess the client and server instances whose execution is going to be tested. The reason for this comes from the fact that the private exponents of all the instances would be unknown to the simulator if we applied the same reduction as in the proof of Lemma 6.2. The problem in the simulation could arise in case the adversary sends the authenticator after making the **Corrupt** query. Therefore, if the simulator makes the right guess, the given random Diffie-Hellman values will be inserted in the instances that are fresh (no **Corrupt** query). The reduction goes as follows.

We construct an algorithm $\mathcal{B}$ that, for given random Diffie-Hellman values $\langle X, Y \rangle$ such that $X \leftarrow g^x$ and $Y \leftarrow g^y$, attempts to solve the CDH assumption (i.e. computes Z such that $Z = DH_g(X, Y)$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{B}$ chooses distinct random indexes $b_1, b_2 \leftarrow \{1, 2, \ldots, n_{se}\}$ and simulates the protocol for $\mathcal{A}$ with the modification of the rule $\mathbf{C1}^{(3)}$ in case of a b_1th **Send**(C^i, **Start**) query and the rule $\mathbf{S1}^{(3)}$ in case of a b_2th **Send**(S^j, (C, E_1, s_1)) query:

⋆ **Rule C1**$^{(4)}$

For the b_1th query choose $s_1 \leftarrow \mathbb{Z}_q$ and set $E_1 := X$. Otherwise, choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := g^{\psi_1}$. In any case, if $(E_1,\ s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

⋆ **Rule S1**$^{(4)}$

For the b_2th query choose $s_2 \leftarrow \mathbb{Z}_q$ and set $E_2 := Y$. Otherwise, choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := g^{\psi_2}$. In any case, if $(E_2,\ s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

After the game ends, for every $H_1(C,\ S,\ s_1,\ s_2,\ E_1, E_2,\ \sigma,\ PW_{C,S})$ query the adversary $\mathcal{A}$ makes, where pairs $(E_1,\ s_1)$ and $(E_2,\ s_2)$ were generated after b_1th and b_2th **Send** query, the password-derived base is correct, and the corresponding $H_0(C, S, pw_C)$ query was made,

$$(\sigma Y^{-s_1} X^{-s_2} g^{-s_1 s_2 \alpha})^{\alpha} \tag{6.12}$$

is added to the list $\mathcal{L}_Z$ of possible values for $DH_g(X, Y)$ of size at most n_{h1}.

From the adversary's view, the simulation $\mathcal{B}$ runs is indistinguishable until the adversary triggers **AskH1-Paired**$_4$. The probability that $\mathcal{B}$ will guess the correct client instance, the correct server instance, and the correct Z value from $\mathcal{L}_Z$ is at least $1/(n_{se}^2 n_{h1})$. The running time of $\mathcal{B}$ is $t' = O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$. Thus, the Lemma 6.3 follows from the fact that the probability of $\mathcal{B}$ solving the CDH assumption is at least $\Pr[\textbf{AskH1-Paired}_4]/(n_{se}^2 n_{h1})$.

□

Before estimating the probability of **AskH1-withS**$_4$ occurring, we evaluate that of **Coll**$_S$, which happens if $\mathcal{A}$ makes two explicit password guesses at the same server instance. Since there are no collisions on H_0 outputs, the only way for $\mathcal{A}$ to accomplish this is if a collision occurs on the first κ-bits of two H_1 queries made by $\mathcal{A}$, with $PW_1 \neq PW_2$. The probability of this occurring is bounded by the birthday paradox $n_{h1}^2/2^{\kappa+1}$.

⋆ **Rule** $H_1^{(4)}$

If $w = (C, S, *, *, *, *, *, PW_{C,S})$, $((C, S, pw_C), PW_{C,S}) \notin \mathcal{L}_{\mathcal{A}}$, and **Corrupted** is false or if **Coll**$_S$ event occurs abort. Otherwise, choose $r \leftarrow \{0,1\}^{3\kappa}$, write the record (w, r) in the list $\mathcal{L}_{h1}$, and answer with r.

Now, without any collision on H_0 and H_1 oracles, each authenticator ϕ coming from $\mathcal{A}$ via a **Send** query corresponds only to one password pw. Therefore,

$$\Pr[\textbf{AskH1-withS}_4] \leq \frac{n_{se}}{N} \ . \tag{6.13}$$

To bound the probability of $\mathbf{AskH1\text{-}withC}_4$, we first bound the probability of $\mathbf{Coll}_C$, which happens if $\mathcal{A}$ makes three implicit password guesses against the same client instance. The following lemma gives such a bound:

Lemma 6.4. *For any $\mathcal{A}$ running in time t that asks at least three bad H_1 queries with distinct values of PW for the same transcript ((C, E_1, s_1), (S, E_2, s_2), ϕ, τ), generated in a communication between $\mathcal{A}$ and C^i, there exists an algorithm $\mathcal{D}$ running in time $t' = O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the DIDH problem:*

$$\Pr[\mathbf{Coll}_C] \leq 2n_{h0}^3 \left(\mathbf{Adv}_{g,\mathbb{G}}^{didh}(\mathcal{D}) + \frac{n_{h1}^3 + 3n_{se}}{2q} \right) \quad . \tag{6.14}$$

Proof. This lemma actually shows that the DIDH assumption prevents the adversary from making more than *two* password guesses per online attempt on the client. We reduce from DIDH as follows.

We construct an algorithm $\mathcal{D}$ that for given a triple $\langle X, Y, Z \rangle$, where $X \leftarrow g^{1/x}$, $Y \leftarrow g^{1/y}$ and $Z \in \mathbb{G}$, attempts to break the DIDH assumption (i.e. determine whether Z is random or $Z = IDH_g(X, Y) = g^{1/(x+y)}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ chooses pair-wise distinct random indexes d_1, d_2, d_3 $\leftarrow \{1, 2, \ldots, n_{h0}\}$, chooses random non-zero exponents u_1, u_2, u_3 in $\mathbb{Z}_q$, and simulates the protocol for $\mathcal{A}$ as follows.

The simulation will be running as in the previous game $\mathbf{G_3}$ until the selected H_0 queries d_1, d_2, or d_3 are made. The simulator will abort the game if the inputs to three selected H_0 queries do *not* satisfy following conditions : (a) the passwords pw_1, pw_2, and pw_3 are pair-wise distinct and different from the correct password pw_C; and (b) the strings (C, S) in all three queries are the same.

If the selected H_0 queries are valid, $\langle X, Y, Z \rangle$ values will be plugged in according to the following rules:

$\star$ **Rule** $H_0^{(4)}$

For the d_1th query set $r := X^{u_1}$. For the d_2th query set $r := Y^{u_2}$. For the d_3th query set $r := Z^{u_3}$. For all three selected queries set $\alpha := \perp$. Otherwise, choose $\alpha \leftarrow \mathbb{Z}_q$ and compute $r := g^{\alpha}$. In any case, write the record (w, r, α) to $\mathcal{L}_{h0}$. If $(*, r, *) \in \mathcal{L}_{\mathcal{A}}$, abort the game.

The prerequisites for the $\mathbf{Coll}_C$ event to occur are: (1) valid d_1th, d_2th, or d_3th H_0 queries are selected by the simulator; (2) a pair (E_1, s_1) is generated by an honest client instance after a $\mathbf{Send}(C^i, \mathbf{Start})$ query; (3) the adversary generated a pair

(E_2, s_2) and made a $\mathbf{Send}(C^i,\ (S, E_2, s_2))$ query, where $\text{Good}(E_2, s_2)$ is true and $E_2 \notin \{X, Y, Z\}$; (4a) for each PW_i, received from the selected H_0 queries, at least one bad $H_1(C, S, s_1, s_2, E_1, E_2, \sigma_i, PW_i)$ query is made for the same transcript, where $i \in \{1, 2, 3\}$. (4b) $\sigma_i \neq 1$.

After the game ends, for every $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ PW_i)$ query $\mathcal{A}$ made, where PW_i is equal to any of the plugged values $\{X^{u_1}, Y^{u_2}, Z^{u_3}\}$, a pair

$$\left(E_2, (\sigma^{\frac{u_i}{\psi_1}} E_2^{\frac{-u_i s_1}{\psi_1}} PW_i^{\frac{-u_i s_1 s_2}{\psi_1}} g^{-u_i s_2})\right) \tag{6.15}$$

is added to the list $\mathcal{L}^i_{bad}$.

So, in the case of an $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ X^{u_1})$ query, by stripping away known values from σ, we may identify a guess at E_2^x and place it in the list $\mathcal{L}^1_{bad}$ together with the E_2 value. Remember that the client instance uses rule $\mathbf{C1}^{(4)}$ to compute E_1, which can be represented with $X^{u_1\psi_1 x}$. In order to extract the $F_1 = E_2^x$ value we do as follows. Since

$$\begin{aligned} \sigma &:= DH_{X^{u_1}}(E_2 X^{u_1 s_2}, E_1 X^{u_1 s_1}) \\ &= E_2^{s_1} X^{u_1 s_1 s_2} E_2^{\frac{\psi_1 x}{u_1}} g^{s_2 \psi_1} \ , \end{aligned} \tag{6.16}$$

we get

$$E_2^x := \sigma^{\frac{u_1}{\psi_1}} E_2^{\frac{-u_1 s_1}{\psi_1}} PW^{\frac{-u_1 s_1 s_2}{\psi_1}} g^{-u_1 s_2} \ . \tag{6.17}$$

The same goes for H_1 queries where the values Y and Z are plugged, in which case the corresponding F_2 and F_3 are computed, respectively. At the end of the simulation, $\mathcal{D}$ checks if for any E_2 value there exist pairs $(E_2, F_1) \in \mathcal{L}^1_{bad}$, $(E_2, F_2) \in \mathcal{L}^2_{bad}$ and $(E_2, F_3) \in \mathcal{L}^3_{bad}$, such that $F_1 F_2 = F_3$. If there exist three such pairs, then $\mathcal{D}$ will output $b' = 1$, and otherwise $b' = 0$.

Now let us analyze the probability that $\mathcal{D}$ returns a correct answer. Suppose first that Z is random. The algorithm $\mathcal{D}$ will return a wrong answer if by chance equation $F_1 F_2 = F_3$ holds. Since the u_i values are random, the probability of this happening is at most n_{h1}^3/q by the union bound. Now suppose that $Z = IDH_g(X, Y)$. The probability of aborting in case E_2 is equal to X, Y or Z is at most $3n_{se}/q$. If the adversary triggers $\mathbf{Coll}_C$, then $\mathcal{D}$ will correctly answer with $b' = 1$ only in case it correctly guessed d_1, d_2, and d_3 from $\{1, 2, \ldots, n_{h0}\}$, which happens with probability

of $1/n_{h0}^3$. Therefore, the probability of $\mathcal{D}$ returning a correct answer is at least

$$\begin{aligned}\Pr[b' = b] &\geq \Pr[b' = 1|b = 1]\Pr[b = 1] + \Pr[b' = 0|b = 0]\Pr[b = 0] \\ &\geq \left(\frac{\Pr[\mathbf{Coll}_C]}{n_{h0}^3} - \frac{3n_{se}}{q}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{n_{h1}^3}{q}\right)\left(\frac{1}{2}\right) \quad . \end{aligned} \tag{6.18}$$

Thus,

$$\Pr[\mathbf{Coll}_C] \leq 2n_{h0}^3 \left(\mathbf{Adv}_{g,\mathbb{G}}^{didh}(\mathcal{D}) + \frac{n_{h1}^3 + 3n_{se}}{2q}\right) \quad . \tag{6.19}$$

From the adversary's view, the simulation $\mathcal{D}$ runs is indistinguishable unless $\mathbf{Coll}_C$ event occurs. The probability of this happening is bounded by (6.19). $\mathcal{D}$'s running time is $t' := O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$ and thus Lemma 6.4 follows. □

Now, without any collision on the H_0 and H_1 outputs, $\mathcal{A}$ impersonating the server to an honest client instance can test at most two passwords per impersonation attempt. Therefore,

$$\Pr[\mathbf{AskH1\text{-}withC}_4] \leq \frac{2n_{se}}{N} \quad . \tag{6.20}$$

Thus,

$$\Pr[\mathbf{AskH1}_4] \leq \frac{3n_{se}}{N} + \frac{n_{h1}^2}{2^{\kappa+1}} + n_{h1}(1 + n_{se}^2)Succ_{PW,\mathbb{G}}^{cdh}(t') + \Pr[\mathbf{Coll}_C] \quad . \tag{6.21}$$

By combining the above equations the bound for semantic security follows. The bound for the mutual authentication is derived in a similar way, by noting that from Def. 3.4 we have $\mathbf{Adv}_{\texttt{dragonfly}}^{ma}(\mathcal{A}) \leq \Pr[\mathbf{Auth}_0^{c2s}] + \Pr[\mathbf{Auth}_0^{s2c}]$.

□

6.5 Efficiency Analysis

The communication and the computation complexity of Dragonfly, as specified in Figure 6.1, are presented in Table 6.1. As pointed out by Clarke and Hao in [34], the Dragonfly protocol is susceptible to an offline dictionary attack in case the group membership validation (Good) is omitted[3]. The cost of such validation is equal to one modular exponentiation in the size of q per user.

As in the case of RO-J-PAKE (see Section 5.6), Dragonfly's "hunting-and-pecking" procedure involves hashing (H_0) into the protocol's underlying group[4]. Thus, the

[3]The client has to validate the ephemeral value E_2, while the server has to check E_1.

[4]Notice that in case of Dragonfly, the server (in some applications also client), may store the value of the password element PW, thus running "hunting-and-pecking" operation only once. This is not possible in case of RO-J-PAKE since the hash is computed over the ephemeral values.

Table 6.1: The efficiency study of Dragonfly

Protocol	Complexity	
	Communication	Computation
Dragonfly	$2 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	$6\|q\|$-bit exp + 2 memb + 2 H_0

same analysis as in 5.6 applies. Table 5.2 is again relevant, since recommended groups for Dragonfly and J-PAKE are the same. Therefore, if $\mathbb{G}$ is deployed as the q-order subgroup of $GF(p)^*$ (SFFs), the cost of single H_0 invocation will be approximately $5|q|$-bit exponentiations when $|p| = 1024$ and $|q| = 160$. This value will grow to around $8|q|$-bit exponentiations when $|p| = 2048$ and $|q| = 224$. This fact brings Dragonfly, in terms of required computation for the first set of parameters, very close to CRS-J-PAKE (18 vs 20 $|q|$-bit exp). As argued in Section 5.6, EC instantiation of H_0 should not be a problem in terms of computation complexity (hashing into elliptic curve is about 16 times cheaper than exponentiating).

Chapter 7

Conclusion

In this chapter, we offer a brief summary of the results included in this thesis and discuss how those results contribute to the discussion on the security of PAKEs. Lastly, we present possible directions for the future study.

7.1 Conclusions

In this thesis, the objects of our interest were Password-Authenticated Key Exchange (PAKE) protocols. Considered as well-studied cryptographic objects in academia, PAKE protocols are just starting to appear more widely as building blocks in commercial real-world applications. They are typically used to generate keys that will grant two (or more) parties means to subsequently establish some type of secure channel between them. More specifically, the secret key derived from a PAKE execution is typically used to authenticate and/or encrypt some data payload using symmetric key protocols.

Composition Result. It is unfortunate that most PAKEs of practical interest, such as [13, 12, 84, 4, 79, 1, 80] (including three protocols analyzed in this thesis), are studied and proven secure in game-based models that do not necessarily provide composition guarantees. Therefore, to substantiate the expected security properties, a new security proof would need to be exhibited for the entire composed protocol from scratch. As a result of Theorem 4.1, a modular design of more complex protocols is possible: One can obtain the secure protocol that would consist of RoR-secure PAKE protocol followed by a secure symmetric key protocol, without any additional analysis[1].

[1]This is not completely true, since one needs to make sure that a public matching algorithm exist for PAKE component. In other words, an eavesdropping adversary should be able to deduce partner

Note that our result applies to any construction that includes an implicitly authenticated PAKE and a key confirmation method which – when composed – satisfy RoR definition. As an example of such composition that is not covered with our composition result, we mention two recently proposed TLS-style PAKEs: TLS-EC-J-PAKE [35] and TLS-Dragonfly [58]. The reason for this is that both of them use the key confirmation method (based on "Finished" message) that does not provide indistinguishable keys and thus is insecure in game-based models (both RoR and FtG).

Looking at our result from a different angle, recall that the RoR model was initially introduced by Abdalla et al. in [3] so that they could prove the security of a three-party PAKE generically constructed from two runs of a two-party PAKE. In addition, they prove that RoR security is better than FtG security for all PAKEs. Our work further validates the latter claim. Indeed, RoR, in PAKE setting, allows us to prove some composability in a way that seems difficult to adapt to FtG.

Protocol Analysis. Although many different protocols that accomplish PAKE have been proposed over the last two decades, only a few newcomers managed to find their way to real world applications - albeit lacking an intense and prolonged public scrutiny. As a step in the direction of providing one, this dissertation has considered the security and efficiency of two relatively recently proposed PAKE protocols - Dragonfly [57] and J-PAKE [54].

In particular, we prove the security of a very close variant of Dragonfly employing the standard Find-then-Guess (FtG) model of security that incorporates forward secrecy, unlike Mackenzie's analysis [83] of SPEKE[2]. Thus, our work confirms that Dragonfly's main flows are sound. The Theorem 6.1's statement indicates that the adversary may successfully guess up to six passwords per send query. Using a much more complex analysis, most likely we could replace the constant 6 in the non-negligible term by 2, and count per instance rather than per send query, but this would be at the cost of readability of the already intricate proof. Also, by virtue of the contents of Lemma 6.4, 2 is certainly the best we could do with this particular analysis.

Furthermore, we contribute to the discussion by proposing and examining two variants of J-PAKE - which we call RO-J-PAKE and CRS-J-PAKE - each making use of two less zero-knowledge proofs than the original protocol, at the cost of an additional security assumption. We highlight the importance of using labels in NIZK

instances just by observing the network. However, as we mentioned before, this property is fulfilled for all relevant PAKEs.

[2]It is highly probable that SPEKE is forward secure as well.

proofs, as they offer a new way to improve the tightness of the reduction[3]. Their function in J-PAKE-style protocols is to link two message rounds in one protocol session together, thus taking away from the adversary one degree of freedom when trying to inject or modify messages. This renders the use of a random oracle for the confirmation code computation in the original J-PAKE [54] obsolete. In terms of efficiency, our work reveals that CRS-J-PAKE has an edge over J-PAKE for both standard group choices: subgroups of finite fields and elliptic curves. The same is true for RO-J-PAKE, but only when instantiated with elliptic curves. We believe our results can be useful to anyone interested in implementing J-PAKE, as perhaps either of these two new protocols may also be options, depending on the deployment context. As a concluding remark regarding Section 5, we stress that EC version of PAKE protocols generally brings significant efficiency gains due to smaller parameters: this includes faster computations and smaller keys. These advantages are especially important in environments such as IoT where processing power, storage space, bandwidth, or power consumption are possibly constrained.

Assumptions. One needs to be aware of shortcomings of the provable security approach in the ideal model (2.5.1) and the trusted setup model (CRS) while looking at these security reduction results. On the bright side, in case of RO-J-PAKE a password is *not* used as an input to a hash function modeled as RO. Therefore, concerns around constant time implementation of such algorithm are not an issue. In some practical scenarios, CRS assumption may not be appropriate since trust must be placed on an entity that will select such public string: in the case of CRS-J-PAKE this is a (random) group element for which discrete logarithm must stay unknown. CRS-J-PAKE is not by any means an exception in this regard: SPAKE2 [4], KOY [71], and GL [46] all use such setup assumption. A possible scenario of CRS-dependent PAKE use would be within a large organization where such PAKE could be utilized as a secure authentication mechanism for employees' login. In this case, the organization would be responsible for CRS generation, hence this public string could be hard-wired into the protocol's implementation.

Note that the proof which indicates that Schnorr signatures [105] are in fact SSE-NIZKPoK (see 2.6.3), assumes *algebraic* adversary in the random oracle model. More generally, those proofs using that sort of the adversary exclude only a class of possible attacks (see Section 2.5.3).

[3]Abdalla et al. [1] were first to include labels in NIZK proofs used in J-PAKE, and we followed their lead. Unfortunately, neither of two internet IETF drafts that provide J-PAKE's specifications [41, 35] include labels in NIZK proofs.

As a final word of caution, there are classes of attacks that game-based security models do not cover, such as various *side channel attacks* (e.g. timing attacks, differential fault analysis, differential power analysis). Nevertheless, the security reduction results are of the great importance since they provide a very good indication that the higher-level protocols do not have significant design flaws. More concretely, they show that no attack on the protocol is possible without exploiting a weakness of the underlying hardness assumptions, idealized primitive or limited adversary.

7.2 Summary of Contributions

Bearing in mind the importance of valid security guarantees, we see our results as a valuable contribution that can help to increase the confidence in the security of PAKE protocols. More specifically, the main contributions of this thesis are:

- The analysis of the compositional features of the game-based PAKE models revealed that PAKE protocols – secure in the sense of the game-based Real-or-Random (RoR) definition of Abdalla et al. [3] – allow for automatic, secure composition with arbitrary, higher-level symmetric key protocols under condition that the public matching algorithm (see 3.3.3) exists for underlying PAKE protocol. However, we emphasize that to the best of our knowledge, for nearly all published PAKEs this is always the case.

- In Section 5, we proposed two new variants of J-PAKE by introducing additional assumptions (RO and CRS) and showed that the security proof from [1] can be adapted to cover our proposals. Also, we compared the overall efficiency of all three protocols when instantiated with ECs or SFFs and concluded that our proposals indeed offer significant improvements in terms of computational effort and communication cost.

- Finally, we proved the security of a very close variant of Dragonfly in the random oracle model. The proof shows in particular that Dragonfly's main flows - a kind of Diffie-Hellman variation with a password-derived base - are sound. In our analysis we employed the Bellare et al. [9] security model, which incorporates forward secrecy.

7.3 Future Research

In recent years PAKE protocols have been getting more attention of security community [104]. Even though PAKEs have been around for quite a long time, the problems from a practical and implementation perspective have not yet been entirely solved. Namely, since some of the computations within PAKE protocols may include low entropy password, side-channel attacks are of great concern. On the other hand, additional theoretical work is required regarding PAKE protocols - especially those considered for a wide-scale use. Therefore, we propose new directions for further research related to PAKE through some open problems:

- As future work, it would be interesting to adapt the study in [25] to the password-based case. This study, among other things, aims to take into account the incorporation of certain specific session-key-dependent messages (such as the "Finished" message from TLS). This would be vital for the protocols recently specified on the IETF [35, 58].

- Another observation worth making is that formally, our composition result could certainly be adapted to the case of any authenticated key exchange. Specifically, the use of the RoR model in order to avoid degradation in Lemma 4.2 could be potentially useful to conduct a study similar to this one or that in [26] for other key exchange authentication mechanisms the security of which - in the game-based setting - can only be guaranteed up to a certain non-negligible value.

- Since RO-J-PAKE using SFFs – the protocol we proposed in Section 5 – is the least efficient when compared with J-PAKE and CRS-J-PAKE due to the application of the hash function H_0, it would be interesting to see if it can be proven secure using a large SFF (and therefore, a "small r"), all while using a short-exponent-type complexity assumption (e.g. as in [86]).

- In the case of Dragonfly, there are several questions that remain open: Firstly, the main flows of the Dragonfly have been integrated into several different protocols/standards [57, 56, 59, 58, 64]. Since specifications of Dragonfly vary across considered protocols, it would be interesting to have a comprehensive analysis of all these standards. The analysis could be modular: each specified variant of Dragonfly would first need to be analyzed using RoR model, after which one could investigate if our composition result from Theorem 4.1 applies to the

higher-level protocol. Secondly, it would helpful to see if the Dragonfly proof that we exhibited can be made tighter. In particular, while it helps readability, the technique of using private oracles as in [21] that we follow in our proof seems less fine-grained than the systematic "backpatching" of, e.g. [84]. Finally, it would be interesting to show if the security of Dragonfly (and SPEKE) could be based on an assumption other than DIDH.

- Following Bader et al. [6], obtaining tighter security bounds for PAKEs would be fruitful, since a loose reduction, in turn, implies larger security parameters.

References

[1] Michel Abdalla, Fabrice Benhamouda, and Philip MacKenzie. Security of the J-PAKE Password-Authenticated Key Exchange Protocol. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 571–587. IEEE Computer Society, 2015.

[2] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-Key Encryption Indistinguishable Under Plaintext-Checkable Attacks. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, volume 9020 of *LNSC*, pages 332–352. Springer, 2015.

[3] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In Serge Vaudenay, editor, *Public-Key Cryptography – PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, 2005.

[4] Michel Abdalla and David Pointcheval. Simple Password-Based Encrypted Key Exchange Protocols. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, 2005.

[5] Elena Andreeva, Bart Mennink, Bart Preneel, and Marjan Škrobot. Security Analysis and Comparison of the SHA-3 Finalists BLAKE, Grøstl,JH, Keccak, and Skein. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012*, volume 7374 of *LNCS*, pages 287–305. Springer, 2012.

[6] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-Secure Authenticated Key Exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC - 2015*, volume 9014 of *LNCS*, pages 629–658. Springer, 2015.

[7] M. Badra. Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode. RFC 5487, RFC Editor, March 2009.

[8] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997*, pages 394–403. IEEE Computer Society, 1997.

[9] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

[10] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO 1993*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.

[11] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[12] Mihir Bellare and Phillip Rogaway. The AuthA Protocol for Password-based Authenticated Key Exchange. In *IEEE P1363*, pages 136–3, 2000.

[13] Steven M. Bellovin and Michael 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.

[14] BlueKrypt. Key Length. http://www.keylength.com/en/4/, 06-04-2017.

[15] Manuel Blum, Paul Feldman, and Silvio Micali. Non-Interactive Zero-Knowledge and Its Applications (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing - STOC 1988*, pages 103–112. ACM, 1988.

[16] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-Interactive Zero-Knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[17] Dan Boneh. The Decision Diffie-Hellman Problem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, 1998*, volume 1423 of *LNCS*, pages 48–63. Springer, 1998.

[18] Joseph Bonneau. *Guessing Human-chosen Secrets*. PhD thesis, University of Cambridge, UK, 2012.

[19] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank 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.

[20] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.

[21] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New Security Results on Encrypted Key Exchange. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography - PKC 2004*, volume 2947 of *LNCS*, pages 145–158. Springer, 2004.

[22] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient Indifferentiable Hashing into Ordinary Elliptic Curves. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, 2010.

[23] Daniel R. L. Brown. Generic Groups, Collision Resistance, and ECDSA. *Des. Codes Cryptography*, 35(1):119–152, 2005.

[24] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Darmstadt University of Technology, 2013.

[25] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: Relaxed yet Composable Security Notions for Key Exchange. *International Journal of Information Security*, 12(4):267–297, 2013.

[26] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway Key Exchange Protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011*, pages 51–62. ACM, 2011.

[27] Ran 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.

[28] Ran Canetti and Marc Fischlin. Universally Composable Commitments. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.

[29] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally Composable Password-Based Key Exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, 2005.

[30] Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[31] Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002.

[32] Céline Chevalier, Stéphanie Delaune, Steve Kremer, and Mark Dermot Ryan. Composition of Password-Based Protocols. *Formal Methods in System Design*, 43(3):369–413, 2013.

[33] T. Clancy and H. Tschofenig. Extensible Authentication Protocol - Generalized Pre-Shared Key (EAP-GPSK) Method. RFC 5433, RFC Editor, February 2009.

[34] Dylan Clarke and Feng Hao. Cryptanalysis of the Dragonfly Key Exchange Protocol. *IET Information Security*, 8(6):283–289, 2014.

[35] R. Cragie and F. Hao. Elliptic Curve J-PAKE Cipher Suites for Transport Layer Security (TLS). https://datatracker.ietf.org/doc/draft-cragie-tls-ecjpake/, 2016.

[36] Ronald Cramer and Victor Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, 2002.

[37] 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.

[38] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–652, 1976.

[39] Jintai Ding, Saed Alsayigh, Jean Lancrenon, Saraswathy RV, and Michael Snook. Provably Secure Password Authenticated Key Exchange Based on RLWE for the Post-Quantum World. *IACR Cryptology ePrint Archive*, 2016:552, 2016.

[40] John Engler, Chris Karlof, Elaine Shi, and Dawn Song. Is it too late for PAKE? In *Web 2.0 Security and Privacy Workshop 2009 (W2SP 2009)*, May 2009.

[41] Hao Feng. J-PAKE: Password Authenticated Key Exchange by Juggling. https://tools.ietf.org/html/draft-hao-jpake-05, 2016.

[42] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO 1986*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[43] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999. http://www.rfc-editor.org/rfc/rfc2616.txt.

[44] OpenSSL Software Foundation. OpenSSL. https://www.openssl.org/, 06-04-2017.

[45] The Mozilla Foundation. Firefox Sync. https://www.mozilla.org/en-US/firefox/sync/, 06-04-2017.

[46] Rosario Gennaro and Yehuda Lindell. A Framework for Password-Based Authenticated Key Exchange. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, 2003.

[47] Oded Goldreich and Yehuda Lindell. Session-Key Generation Using Human Passwords Only. In Joe Kilian, editor, *Advances in Cryptology CRYPTO 2001*, volume 2139 of *LNCS*, pages 408–432. Springer, 2001.

[48] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2)/ 270-299, 1984.

[49] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[50] Jens Groth. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, 2006.

[51] Jens Groth and Rafail Ostrovsky. Cryptography in the Multi-string Model. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, 2007.

[52] Thread Group. Thread Protocol. http://threadgroup.org/, 06-04-2017.

[53] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Science & Business Media, 2006.

[54] Feng Hao and Peter Ryan. J-PAKE: Authenticated Key Exchange without PKI. *Transactions on Computational Science*, 11:192–206, 2010.

[55] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/rfc6749.txt.

[56] D. Harkins. Secure Pre-Shared Key (PSK) Authentication for the Internet Key Exchange Protocol (IKE). RFC 6617, RFC Editor, June 2012.

[57] D. Harkins. Dragonfly Key Exchange. RFC 7664, RFC Editor, November 2015.

[58] D. Harkins. Secure Password Ciphersuites for Transport Layer Security (TLS). https://datatracker.ietf.org/doc/draft-harkins-tls-dragonfly/, 2016.

[59] D. Harkins and G. Zorn. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. RFC 5931, RFC Editor, August 2010.

[60] Dan Harkins. Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks. In *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, SENSORCOMM '08, pages 839–844. IEEE Computer Society, 2008.

[61] Dan Harkins. Dragonfly Key Exchange. https://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/, 2015.

[62] Cormac Herley and Paul C. van Oorschot. A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012.

[63] Thomas Icart. How to Hash into Elliptic Curves. In *Advances in Cryptology - CRYPTO 2009*, pages 303–316, 2009.

[64] 802.11-2012 - IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Standard, IEEE Standards Association, Piscataway, NJ, USA, 2002.

[65] Standard Specifications for Password-based Public Key Cryptographic Techniques. Standard, IEEE Standards Association, Piscataway, NJ, USA, 2002.

[66] 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.

[67] David P Jablon. Strong Password-Only Authenticated Key Exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[68] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the Security of TLS-DHE in the Standard Model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, 2012.

[69] Shaoquan Jiang and Guang Gong. Password Based Key Exchange with Mutual Authentication. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, volume 3357 of *LNCS*, pages 267–279. Springer, 2004.

[70] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[71] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, 2001.

[72] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Forward Secrecy in Password-Only Key Exchange Protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks – SCN 2002*, volume 2576 of *LNCS*, pages 29–44. Springer, 2002.

[73] Jonathan Katz and Vinod Vaikuntanathan. Round-Optimal Password-Based Authenticated Key Exchange. In Yuval Ishai, editor, *Theory of Cryptography - TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, 2011.

[74] Franziskus Kiefer. *Advancements in Password-based Cryptography*. PhD thesis, University of Surrey UK, 2016.

[75] Neal Koblitz and Alfred J. Menezes. The Random Oracle Model: A Twenty-year Retrospective. *Designs, Codes and Cryptography*, 77(2-3):587–610, 2015.

[76] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, 2010.

[77] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the Security of the TLS Protocol: A Systematic Analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8042 of *LNCS*, pages 429–448. Springer, 2013.

[78] Taekyoung Kwon. Practical Authenticated Key Agreement Using Passwords. In Kan Zhang and Yuliang Zheng, editors, *Information Security*, volume 3225 of *LNCS*, pages 1–12. Springer, 2004.

[79] Jean Lancrenon and Marjan Škrobot. On the Provable Security of the Dragonfly Protocol. In Javier Lopez and Chris J. Mitchell, editors, *Information Security – ISC 2015*, volume 9290 of *LNCS*, pages 244–261. Springer, 2015.

[80] Jean Lancrenon, Marjan Škrobot, and Qiang Tang. Two More Efficient Variants of the J-PAKE Protocol. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *Applied Cryptography and Network Security – ACNS 2016*, volume 9696 of *LNCS*, pages 58–76. Springer, 2016.

[81] Brian Lloyd and William Allen Simpson. PPP Authentication Protocols. RFC 1334, RFC Editor, October 1992. http://www.rfc-editor.org/rfc/rfc1334.txt.

[82] BuiltWith Pty Ltd. Ssl-by-default. https://trends.builtwith.com/ssl/SSL-by-Default, 06-04-2017.

[83] Philip 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.

[84] Philip MacKenzie. The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46, 2002.

[85] Philip D. MacKenzie. More Efficient Password-Authenticated Key Exchange. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001*, volume 2020 of *LNCS*, pages 361–377. Springer, 2001.

[86] Philip D. MacKenzie and Sarvar Patel. Hard Bits of the Discrete Log with Applications to Password Authentication. In *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *LNCS*, pages 209–226. Springer, 2005.

[87] Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-Authenticated Key Exchange Based on RSA. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 599–613. Springer, 2000.

[88] Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-Authenticated Key Exchange based on RSA. *Int. J. Inf. Sec.*, 9(6):387–410, 2010.

[89] Mark Manulis, Douglas Stebila, Franziskus Kiefer, and Nick Denham. Secure Modular Password Authentication for the Web using Channel Bindings. *Int. J. Inf. Sec.*, 15(6):597–620, 2016.

[90] Ueli M. Maurer. Abstract Models of Computation in Cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference*, volume 3796 of *LNCS*, pages 1–12. Springer, 2005.

[91] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, CA, USA, 1979.

[92] Tyler Moore and Richard Clayton. Examining the Impact of Website Take-down on Phishing. In Lorrie Faith Cranor, editor, *Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit 2007*, volume 269 of *ACM International Conference Proceeding Series*, pages 1–13. ACM, 2007.

[93] Robert Morris and Ken Thompson. Password Security - A Case History. *Commun. ACM*, 22(11):594–597, 1979.

[94] 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.

[95] Pascal Paillier and Damien Vergnaud. Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20. Springer, 2005.

[96] David Pointcheval. Password-Based Authenticated Key Exchange. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012*, volume 7293 of *LNCS*, pages 390–397. Springer, 2012.

[97] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT 1996*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[98] Moonchild Productions. Pale Moon. https://www.palemoon.org/sync/privacy.shtml, 06-04-2017.

[99] David Recordon and Drummond Reed. OpenID 2.0: a platform for user-centric identity management. In Ari Juels, Marianne Winslett, and Atsuhiro Goto, editors, *Proceedings of the 2006 Workshop on Digital Identity Management*, pages 11–16. ACM, 2006.

[100] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/draft-ietf-tls-tls13-18, 2016.

[101] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[102] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.

[103] Allan Lee Scherr. An Analysis of Time-shared Computer Systems.

[104] Joern-Marc Schmidt. Requirements for PAKE schemes. https://tools.ietf.org/html/draft-irtf-cfrg-pake-reqs-07, 2016.

[105] Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO 1989*, volume 435 of *LNCS*, pages 239–252. Springer, 1989.

[106] Andrew Shallue and Christiaan van de Woestijne. Construction of Rational Points on Elliptic Curves over Finite Fields. In *Algorithmic Number Theory, 7th International Symposium, ANTS-VII*, pages 510–524, 2006.

[107] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Security Proof of Augpake. Cryptology ePrint Archive, Report 2010/334, 2010. http://eprint.iacr.org/2010/334.

[108] Victor Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

[109] Victor Shoup. On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012, 1999. http://eprint.iacr.org/1999/012.

[110] W. Simpson. The Point-to-Point Protocol (PPP). STD 51, RFC Editor, July 1994.

[111] William Allen Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, RFC Editor, August 1996. http://www.rfc-editor.org/rfc/rfc1994.txt.

[112] Marjan Škrobot. Provable Security Analysis of SHA-3 Candidates. Master's thesis, University of Novi Sad, 2012.

[113] Frank Stajano. Pico: No More Passwords! In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Frank Stajano, editors, *Security Protocols XIX*, volume 7114 of *LNCS*, pages 49–81. Springer, 2011.

[114] Brian Warner. Magic Wormhole. https://github.com/warner/magic-wormhole, 06-04-2017.

[115] Stephen C. Williams. *On the Security of Key Exchange Protocols*. PhD thesis, University of Bristol, UK, 2011.

[116] Thomas D. Wu. The Secure Remote Password Protocol. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1998*. The Internet Society, 1998.

Appendix A

Experiment Source Codes from Chapter 5

```
//  ./tests/TestAddMul_x 80 23 7
#define N_TESTS 1
//#define DEBUG 1

//stringstream
#include <sstream>

#include "Plaintext.h"
#include "DoubleCRT.h"
#include "Ciphertext.h"
#include "NTL/ZZ_pX.h"
#include <NTL/vector.h>

#include "SHA1.hpp"

#include <time.h>
#include "FHE-SI.h"
#include <vector>

//time cost
#include <ctime>
std::clock_t start;
double duration;

//ns time test
#include <chrono>
typedef std::chrono::high_resolution_clock Clock;

#include <NTL/ZZ_p.h>
#include <NTL/ZZ.h>
//http://www.shoup.net/ntl/doc/ZZ.cpp.html
//http://www.shoup.net/ntl/doc/ZZ_p.cpp.html

int len=256;
int TIMES=1000;

ZZ p;
ZZ_p a, b;
ZZ_p Px, Py;

void CurveGen(){
        //ZZ p,
        //ZZ_p a, b;

        //gen p
```

```
        GenPrime(p, len, 80); //prime with 2^(-80)
        while(rem(p,3)!=2)
                GenPrime(p, len, 80);
        //cout<<"p:"<<hex<<p<<endl;

        ZZ_p::init(p);

        //gen a, b
        random(a);
        random(b);

        //gen a random point, not neccecarily on the curve
        //From P12 "Generation Methods of Elliptic Curves"
        //the whole computation of G and verifying its order
    //is of bit-complexity O(log^4(p))
        random(Px);
        random(Py);
}

void Pgen1(ZZ m){
        ZZ tp3;
        ZZ_p u, v, x, y;
        ZZ_p tp1, tp2, tp4;

        //u=hash(m)
        //sha-1
        //https://github.com/jasinb/sha1/blob/master/main.c
        conv(u,m);

        //embed message
        //compute v
        mul(tp1,a,3);
        power(tp2, u, 4);
        sub(tp1,tp1,tp2);
        mul(tp2, u, 6);
        div(v, tp1, tp2);

        power(tp1, v, 2);
        sub(tp1, tp1, b);
        mul(tp2, u, u); // u^6/27=(u^2/3)^3
        div(tp2, tp2, 3); //tp2=u^2/3
        power(tp4, tp2, 3);
        sub(tp1, tp1, tp4);

        mul(tp3, p, 2);
        sub(tp3, tp3, 1);
        div(tp3, tp3, 3);

        power(tp1, tp1, tp3);
        add(x, tp1, tp2);  //x

        mul(tp1, u, x);
        add(y, tp1, v);  //y

        //cout<<"x:"<<hex<<x<<endl;
        //cout<<"y:"<<hex<<y<<endl;
}

//point addition in Jacobian
void AD(ZZ_p& X1,ZZ_p& Y1,ZZ_p& Z1,ZZ_p X2,ZZ_p Y2){
        ZZ_p A, B, C, D, E, F, G;
        ZZ_p Z2;

        conv(Z2,1);

        A=(X1*power(Z2,2));
        B=(X2*power(Z1,2));
        C=(Y1*power(Z2,3));
```

```
        D=(Y2*power(Z1,3));
        E=(B-A);
        F=(D-C);

        G=power(Z2,2);

        X1=(-power(E,3)-2*A*power(E,2)+power(F,2)) ;
        Y1=(-C*power(E,3)+F*(A*power(E,2)-X1)) ;
        Z1=(Z1*Z2*E) ;
}

//point doubling in Jacobian
void DB(ZZ_p& X1,ZZ_p& Y1,ZZ_p& Z1){
        ZZ_p A, B;

        A=(4*X1*power(Y1,2)) ;
        B=(3*power(X1,2)+a*power(Z1,4)) ;

        Z1=(2*Y1*Z1) ;
        X1=(-2*A+power(B,2)) ;
        Y1=(-8*power(Y1,4)+B*(A-X1)) ;
}

//using m*P for point hashing
//using Jacobian coordination
//if sha-1, m is 160-bit?
void Pgen2(ZZ m){
        ZZ_p X1, Y1, Z1;
        ZZ_p tp1;

        X1=Px;//suppose MSB of m=1
        Y1=Py;
        conv(Z1, 1);
        for(int i=len-1; i>=0; i--)
        {
                DB(X1, Y1, Z1);
                if(bit(m,0)) //LSB
                        AD(X1,Y1,Z1,Px,Py);
                m=m>>1;
        }
        tp1=inv(power(Z1,3));
        Y1=Y1*tp1;
        X1=X1*Z1*tp1;

        //cout<<"X1:"<<hex<<X1<<endl;
        //cout<<"Y1:"<<hex<<Y1<<endl;
}


ZZ stringToNumber(string str)
{
    ZZ number = conv<ZZ>(str[0]);
    long len = str.length();
    for(long i = 1; i < len; i++)
    {
        number *= 16;
        number += conv<ZZ>(str[i]);
    }

    return number;
}

int main(int argc, char *argv[]) {
  ZZ tpm;
  vector<ZZ> m;
  SetSeed(to_ZZ(time(0)));

  //Curve Gen
```

```
  CurveGen();

  //Message Gen1
  for (int i = 0; i < TIMES; i++)
  {
        RandomBits(tpm, len);
        m.push_back(tpm);
  }


  //hash 1
  //"How to Hash into Elliptic Curves"
  start = std::clock();

  for (int i = 0; i < TIMES; i++)
  {
        Pgen1(m.back());
        m.pop_back();
  }

  duration = ( std::clock() - start ) / (double) CLOCKS_PER_SEC;
  cout<<"HASH1:"<< duration/TIMES <<'\n';

  //Message Gen2
  for (int i = 0; i < TIMES; i++)
  {
        RandomBits(tpm, len);
        m.push_back(tpm);
  }

  //hash 2
  //m*P
  start = std::clock();

  for (int i = 0; i < TIMES; i++)
  {
        Pgen2(m.back());
        m.pop_back();
  }

  duration = ( std::clock() - start ) / (double) CLOCKS_PER_SEC;
  cout<<"HASH2:"<< duration/TIMES <<'\n';
}
```