NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DIOGO SPENCER SALOMÃO RAMOS

BSc in Computer Science and Engineering

INVESTIGATING KEY ROTATION SECURITY IN OBLIVIOUS PSEUDORANDOM FUNCTION PROTOCOLS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

INVESTIGATING KEY ROTATION SECURITY IN OBLIVIOUS PSEUDORANDOM FUNCTION PROTOCOLS

DIOGO SPENCER SALOMÃO RAMOS

BSc in Computer Science and Engineering

Adviser: Alexander Davidson
Assistant Professor, NOVA University Lisbon

Co-adviser: João Ribeiro
Assistant Professor, Instituto Superior Técnico, Universidade de Lisboa

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

Investigating Key Rotation Security in Oblivious Pseudorandom Function Protocols

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.0.2) [84].

Acknowledgements

Firstly, I would like to express my gratitude to my adviser Professor Alexander Davidson and my co-adviser Professor João Ribeiro for all their help. I am grateful for their patience and support throughout the development of this thesis. I wish them both continued success and fulfillment in their professional and personal lives.

I would also like to express my deep gratitude to the Department of Computer Science at NOVA School of Science and Technology and all the professors who, over the years, shared their knowledge with me.

I also want to thank my friends and colleagues for all the moments we shared together and all that they have taught me, I will remember the adventures we lived together fondly.

Lastly, I want to express my deepest regards to my family for all the patience and for providing me with all the support that I needed. A special thanks to my mother and my girlfriend, who were always there for me through the highs and lows. Their unwavering support and motivation have been invaluable to me.

Abstract

Over the last years, oblivious pseudorandom functions (OPRFs) have emerged as a fundamental building block in various cryptographic protocols and privacy-preserving technologies. The demand for secure communication protocols has grown exponentially, particularly in cloud-based systems where the transmission of sensitive data is common.

These type of protocols haves gained attention due to their promising approach to preserving user privacy and preventing server-side database intrusions, by using OPRFs to allow secure multiparty computation scenarios where the server and the client don't learn about each others private data. The security of these protocols however is heavily reliant on the strength of the keys generated by the server.

This thesis aims to investigate the potential consequences of the server choosing (intentionally or not) a weak key for the protocol and the changes in the security guarantees when that happens. Furthermore, we analyze if a client selecting a strong password can mitigate the security risks arising from predictable server-side keys, thereby maintaining a secure connection.

With the results of this study, we intend to contribute to a broader understanding of security protocols that use OPRFs and to better understand the impact that weak keys can have in the security of OPRFs protocols, shedding light on the implications of weak keys and the potential mitigation through client password strength. This would significantly improve the security guarantees of many applications.

Keywords: Cryptography, Key Rotation, Oblivious Pseudorandom Functions (OPRFs), Pseudorandom Functions (PRFs), Pseudorandomness, Weak Keys

Resumo

Nos últimos anos, *oblivious pseudorandom functions* (OPRFs) tornaram-se peças fundamentais em vários protocolos criptográficos e tecnologias de preservação de privacidade. A procura de protocolos de comunicação seguros tem crescido exponencialmente, particularmente em sistemas de *cloud*, onde a transmissão de dados sensíveis é frequente.

Estes tipos de protocolos tem ganho atenção devido à sua abordagem promissora para preservar a privacidade dos utilizadores e prevenir intrusões em bases de dados no lado do servidor, utilizando OPRFs para permitir cenários de computação multipartidária segura, onde o servidor e o cliente não tomam conhecimento dos dados privados um do outro. No entanto, a segurança destes protocolos depende fortemente da robustez das chaves geradas pelo servidor.

Esta tese tem como objetivo investigar as potenciais consequências de o servidor escolher (intencionalmente ou não) uma chave de robustez fraca para o protocolo e as alterações nas garantias de segurança quando isso acontece. Além disso, analisamos se um cliente que selecionar uma senha forte pode mitigar os riscos de segurança decorrentes de chaves previsíveis do lado do servidor, mantendo assim uma ligação segura.

Com os resultados deste estudo, pretendemos contribuir para uma compreensão mais ampla dos protocolos de segurança que utilizam OPRFs e entender melhor o impacto que chaves fracas podem ter na segurança de protocolos OPRF, destacando as implicações de chaves fracas e o potencial de mitigação através da força das senhas dos clientes. Isto melhoraria significativamente as garantias de segurança de muitas aplicações.

Palavras-chave: Criptografia, Rotação De Chaves, *Oblivious Pseudorandom Functions (OPRFs)*, *Pseudorandom Functions (PRFs)*, *Pseudorandomness*, Chaves fracas

Contents

List of Figures

1 Introduction

1.1 Context

In recent years, the proliferation of online services and the exponential growth of digital data have highlighted the critical importance of privacy and security in our digital interactions. As individuals and organizations increasingly rely on cryptographic protocols to protect sensitive information, the study and improvement of modern cryptography and of these protocols has become imperative. One such cryptographic construct that has garnered significant attention is the Oblivious Pseudorandom Function (OPRF), which plays a very important role in various privacy-enhancing technologies [3, 18, 32, 42, 57, 70]. OPRFs enable a client to compute a function on their input with the help of a server without the server learning anything about the client's input. The output of the function is pseudorandom, ensuring privacy and security during the computation. However, as with any cryptographic tool, ensuring the long-term security of OPRF protocols remains a challenge. One area of particular interest is the role of cryptographic keys in OPRF protocols, which are crucial for ensuring security and privacy in modern cryptographic systems.

Because of this, in OPRF protocols as is the case with all cryptographic protocols, care must be taken to ensure that the key that is in use is rotated frequently to ensure that a key compromise event does not cause security breaches for any protocol participants. Such key rotations involve the OPRF server internally deriving a new key randomly. This thesis aims to investigate what can happen if a server gets corrupted and can no longer derive truly random keys, with this investigation we have the objective of contributing to a broader understanding of OPRFs by investigating the impact in security of a non-uniform key being used in an OPRF protocol and what that entails for applications that use OPRFs. Addressing the challenges posed by malicious servers and weak keys remains an open problem in the context of OPRFs [32]. This thesis aims to contribute to this ongoing discussion by exploring the impact of weak keys in an OPRF protocol and potential mitigation strategies.

As the title suggests, the focal point of this thesis revolves around an in-depth analysis

of specific cryptographic protocols and primitives, with a primary emphasis on oblivious pseudorandom functions. To achieve this, we will be following the principles and common practices of modern cryptography.

Consequently, it becomes important to lay the groundwork by providing a comprehensive context for modern cryptography, elucidating its significance and its main principles. Subsequently, we will introduce OPRFs and delve into their importance as a cryptographic mechanism as well as their practical implications in the real world.

1.1.1 Cryptography and Modern Cryptography

Cryptography, was once regarded as simply the encoding and decoding of messages, initially conceived as a tool for securing secret communications, it found its primary application in government and military settings.

In the latter half of the 20th century, cryptography underwent a profound paradigm shift, marked by the emergence of modern (post-80's) cryptography. This new era, characterized by its emphasis on precise definitions, formal assumptions, and rigorous security proofs, signaled a departure from the ad-hoc methods of classical cryptography. As articulated by Katz and Lindell [74], modern cryptography can be described as the field of science that focuses on the examination of mathematical techniques to protect digital information, systems, and distributed computations against adversarial attacks.

With the dawn of the digital age, cryptography evolved into a sophisticated science and emerged as a pivotal component of modern information security, it has become a central topic within computer science allowing us to fortify the foundations of trust, privacy and integrity in our digital world by bridging the gap between mathematical abstraction and practical application.

In current times, we can find the hand of cryptography everywhere as our lives become increasingly intertwined with technology, from personal communications [45, 81], to electronic voting [95], to online financial transactions [35], to the use of VPNs [54] and blockchain technology and digital cash [87, 88], as well as in matters of national security [90, 92], the need for robust cryptographic security measures and privacy safeguards has never been more pronounced.

1.1.1.1 Importance of Modern Cryptography

In today's digital world, where our personal, financial, and professional activities are increasingly conducted online, the significance of modern cryptography cannot be overstated. It plays a vital role in protecting sensitive information during transmission and storage, it provides us with the foundation for secure communication and transaction over the Internet, it allows us to ensure privacy and confidentiality by encoding messages so that only authorized parties can understand them, and it allows us to ensure that data remains unaltered and authentic by using cryptographic techniques that enable us to verify the integrity of messages and confirm their origin.

However, while practical applications might seem the most appealing part of cryptography, it is the rigorous exploration of theory, definitions and proofs in modern cryptography that truly supports cryptographic reliability. By delineating the boundaries of security and formulating provable security guarantees, theoretical constructs empower cryptographers to evaluate the robustness of cryptographic primitives and identify vulnerabilities before they can be exploited in practice.

Cryptographers and researchers scrutinize the theoretical foundations of cryptographic primitives and protocols that are used in real life and establish precise definitions and proofs, providing assurances of security that allow us to trust in the security, integrity and effectiveness of cryptographic constructs in real-world scenarios.

To better understand the importance of the theory and scrutiny of cryptography, in the following section we will give an introduction into what Katz and Lindell [74] identified as the three main principles of modern cryptography: formal definitions, precise assumptions and proofs of security. It is important to note that these principles of modern cryptography are not only relevant to the "theory of cryptography" community. Nowadays, rigorous proofs of security have become a requirement for cryptographic schemes to be standardized and the importance of these principles is widely understood by developers and security engineers who use cryptographic tools to build secure systems.

1.1.1.2 Principles of Modern Cryptography

In today's context of modern cryptography, cryptographic schemes and constructions are analyzed in a systematic manner and have the objective of giving rigorous proof that they are secure. This approach is underpinned by three main principles: formal definitions, precise assumptions and proofs of security.

In this subsection, we will provide a brief overview of these principles (we will delve deeper into these principles and explore them in greater detail in the Related Work, Section 2.1):

- **Formal definitions:** Modern cryptography emphasizes the importance of establishing formal definitions of security as a fundamental prerequisite in the design of cryptographic primitives and protocols since these definitions provide a clear understanding of the security objectives that they aim to achieve. This is very important because without a clear understanding of the desired security objectives, it becomes challenging to determine whether those objectives have been met.

  Furthermore, clearly defined definitions and security objectives pave the way for a comprehensive analysis providing us with a rigorous framework for security evaluation. They also foster confidence in the security of cryptographic solutions, and help us find out potential vulnerabilities when those definitions are challenged by new threats or advacemetns in cryptanalysis.

- **Precise assumptions:** In modern cryptography precise assumptions are very important since many cryptographic constructions cannot be proven secure unconditionally. Instead their security often relies on widely accepted assumptions regarding the computational capabilities of potential adversaries, it is important to note that these assumptions are carefully chosen based on our understanding of computational hardness and the current state of technology.

  The modern cryptographic methodology necessitates explicit and unequivocal declaration of any such assumption.

  A good example of this is the RSA algorithm [102], a popular cryptographic scheme whose security depends on the assumption that it is computationally infeasible to factor large numbers into their prime factors efficiently.

  Following this principle is important in modern cryptographic methodology and any such assumption should be explicit. By relying on precise assumption and following this methodology we are able to construct cryptographic systems that are secure under specific computational scenarios.

- **Proofs of security:** This third principle is built on the idea of the two previous principles, that if we rely on precise definitions of security and well defined cryptographic assumptions we can prove the security of cryptographic schemes and constructions and provide formal proofs of security.

  The idea of this principle is crucial and cannot be overemphasized as it defines the vital role of modern cryptography in today's world. In the past, cryptographic schemes were created mainly on the fly and were considered secure if the creators themselves could not identify any vulnerabilities. On the other hand, modern cryptography advocates the design of schemes supported by formal, mathematical proofs of security in well-defined security models.

  These proofs usually involve reduction arguments, where the security of a cryptographic construction or scheme is reduced to the hardness of a well-studied mathematical problem. By showing that breaking the cryptographic scheme implies solving the underlying mathematical problem, we can obtain schemes and constructions that are extremely unlikely to be broken.

  This way we can guarantee that such cryptographic schemes and constructions are secure(unless the the security definitions did not appropriately model the real world security concerns or that the underlying assumptions are false), and by providing proofs of security, we can provide strong evidence that a given scheme or construction is robust against various attacks, presenting a level of trust regarding their practical application.

In this thesis we will abide by these three main principles of modern cryptography, we will present security models with formal definitions, precise assumptions and proofs

of security. This comprehensive approach ensures the validity and reliability of our findings, ultimately offering valuable insights for building more robust and trustworthy cryptographic systems.

1.1.2 Oblivious Pseudorandom Functions

Now that we have established the value of cryptographic analysis and research and the importance of modern cryptography, it is fitting to provide an introduction to primary cryptographic construct that will be analyzing in this thesis: Oblivious Pseudorandom Functions.

In Section 2.4.2, we will go into more detail about what is an OPRF, but in essence an OPRF is a cryptographic construct that allows one party (the client) to obtain the evaluation of a pseudorandom function (PRF) on their input, with the help of another party (the server), without the server learning the client's input and without the client learning the server's secret key. This property makes OPRFs very valuable in privacy-preserving computation, especially in scenarios where sensitive data needs to be processed securely without disclosing it to other parties.

More intuitevlty, a OPRF is a protocol between a client C and a server S where C holds an input x and S holds the key k for some PRF. A PRF is a very common cryptographic primitive that essentialy takes a key and some input and outputs some cryptographically random value (we will go into a more careful explanation in Section 2.3.2). The security goal of the OPRF protocol is that the client C receives the output $y = F(k, x)$ without learning the key k and the server S does not learn the client's input x.

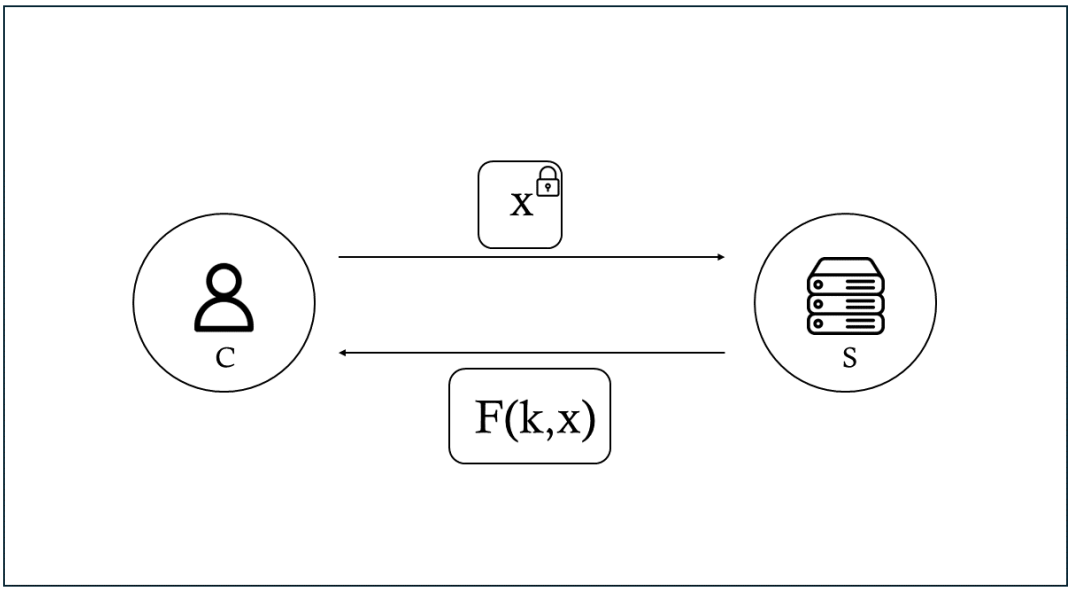

Figure 1.1: OPRF protocol diagram

1.1.2.1 Value of OPRFs

In an increasingly interconnected world, the need for secure and privacy-preserving communication has become paramount and oblivious pseudorandom functions have gained prominence as a powerful cryptographic tool to address these concerns. Specially nowadays when we are living in an era where digital privacy is hard to maintain and our personal data is often utilized by various entities without our explicit consent, OPRFs have become extremely interesting and valuable to us since their obliviousness can allow the construction of privacy-preserving protocols that guarantee that our data remains private.

By allowing parties to perform computations on private data without revealing sensitive information, OPRFs offer a crucial building block for enabling privacy-enhancing technologies such as private set intersection (PSI) [64, 75, 77], secure password-based authentication [69] and anonymous credential systems [43, 44] to name a few. Their ability to facilitate secure data exchange while preserving confidentiality makes OPRFs an important tool in modern cryptography. The adoption and integration of OPRFs into diverse applications and services offer a promising avenue for reinforcing digital privacy and instilling trust in our increasingly interconnected digital world.

Recognizing this importance and the potential impact of OPRFs, standardization bodies and cryptographic communities have started efforts to develop standardized OPRF protocols. Notably, the Internet Engineering Task Force (IETF) has established a working group dedicated to standardizing OPRFs, further underscoring their importance. [42]

These standardization efforts aim to provide a common framework and interoperability for OPRF implementations, fostering adoption, and enabling seamless integration of OPRFs into various cryptographic systems. Standardization ensures that OPRFs meet stringent security requirements and encourages widespread deployment, thereby enhancing trust and promoting consistent best practices.

1.1.2.2 Applications of OPRFs

Oblivious pseudorandom functions are a powerful tool since they allow a client to compute a high-entropy cryptographic object (e.g., a key, or a token) from a low-entropy input (e.g., a username, a an identifier, a password, or a file) and a high-entropy key. The fact that the computation is assisted by one or many servers allows for protocols that are lightweight on the client side. This allows for the secure storage of cryptographic material on servers, which can then be accessed with the assistance of the OPRF. Additionally, the obliviousness of the PRF evaluation conceals the client's protocol input. This, combined with efficiency and robust security, positions OPRFs as one of the most promising tools for enhancing privacy in recent years.

Casacuberta et al. [32] demonstrated that applications of OPRFs in the literature leverage OPRFs in essentially two ways:

1. When OPRFs are used to let clients (re-)compute high-entropy cryptographic objects, such as cryptographic keys. This can be useful for example for: Secure password verification [8, 27, 41, 69], server-assisted encryption [28], secret key recovery/password-encrypted backups [7, 25, 66].

2. When OPRFs are used instead of hash functions, to enforce interaction when computing hash values. This is useful in settings where limitation of hash evaluation is desirable, for example for: precomputation-resistant password hardening [69], rate limiting for web-services [43], secure comparison of private inputs [55, 64] (e.g., contact tracing [73]).

To be more concrete and to show some practical examples of the use of oblivious pseudorandom functions we will mention below some interesting examples of cryptographic protocols that utilize oblivious pseudorandom functions.

One example is secure multiparty computation (MPC). MPC protocols enable multiple parties to jointly compute a function over their private inputs while revealing only the desired output. By allowing parties to securely evaluate functions on private inputs without disclosing sensitive information to each other, OPRFs can help enable secure multiparty computation [72].

Another use that is relevant to look at is private information retrieval (PIR). PIR protocols enable a client to retrieve specific data from a server without revealing which data is being accessed, PIR protocols use OPRFs to ensure that the server learns nothing about the client's query beyond the requested data, preserving user privacy in data retrieval scenarios [57].

Adding to its versatility of uses, OPRFs are instrumental in addressing another significant cryptographic challenge: Private Set Intersection (PSI) [33, 75, 77]. PSI protocols are cryptographic protocols that enable two parties, each holding a set of items, to determine which items are common to both sets without disclosing any other information about the items in their sets. This is used, for example, in Google's Password Checkup [99], a Google service that utilizes PSI to securely check if a user's passwords have been compromised in data breaches without the need to expose the passwords themselves.

OPRFs are also used for password-authenticated key exchange (PAKE) [21, 62] protocols. An example of one of these protocols is OPAQUE [69] a protocol that uses OPRFs to protect user passwords even if the server is compromised or if there are offline attacks. OPAQUE is a cryptographic protocol designed to enhance the security of password-based authentication, it provides password-authenticated key exchange functionality, allowing two parties to establish a secure shared key using only a password as a shared secret. It is an oblivious protocol in the sense that it ensures that no information about the passwords is leaked during the authentication process.

Recently OPRFs have been applied to back up encrypted chat histories in WhatsApp [45] and Facebook Messenger [81].

To conclude, OPRFs have shown to be a central primitive for building secure and privacy-preserving cryptographic protocols and applications in modern computing environments. Their unique properties, such as obliviousness, make them usefull in our everyday life, enabling to construct protocols that protect private data, such as our passwords, our search inputs, our identities and our digital footprints . It is important to note that OPRFs are used in a lot more cases besides the ones we mentioned here and we just gave some examples of their use to give some context and to show their usefulness, we will talk about more applications of OPRFs in Chapter 2.

1.2 Motivation

1.2.1 The Danger of Weak Keys

In an era where digital communication, financial transactions, and personal information rely heavily on cryptography, the security of cryptographic keys is extremely important in our lives. Cryptographic systems support a wide range of services, ensuring confidentiality, authenticity, and integrity of data. However, the strength of these systems is critically dependent on the robustness of the cryptographic keys. Weak keys pose a significant risk, potentially undermining the entire cryptographic infrastructure.

Weak keys in cryptographic systems are those that fail to provide adequate security against adversarial attacks. This may arise due to insufficient randomness (often measured as entropy) during key generation, flawed algorithms, inadequate key lengths, or structural weaknesses in the key generation process. Structural weaknesses can include biases or predictable patterns in key generation that make certain keys more likely to be selected, reducing the effective key space. Weak keys are a security problem since they can be efficiently computed, predicted, or compromised using various cryptanalytic techniques, making them highly vulnerable to attacks.

When weak keys are generated, the entire security of a cryptographic system can be undermined, since weak keys drastically reduce the complexity of brute force and cryptanalytic attacks, making them feasible for even moderately resourced attackers, allowing them to potentially decrypt sensitive data and to forge signatures or authentication tokens.

A well-known real-world example of weak key generation is the Debian OpenSSL vulnerability (2008) [111]. In this incident, a misconfiguration in the Debian Linux distribution's implementation of OpenSSL led to the generation of cryptographic keys that were easily predictable. This issue arose from a bugfix in the OpenSSL library that inadvertently reduced the entropy available during key generation. As a result of this reduction in randomness, keypairs created on affected machines lacked sufficient entropy, making them vulnerable to brute-force attacks. The flaw particularly affected SSL/TLS and SSH servers, where predictable keys could allow attackers to derive the private key. With this information, attackers could impersonate legitimate servers and, in many cases, decrypt secure communication without detection.

1.2.2 Investigating Key Security

As we described in the previous section OPRFs protocols find application in various privacy-preserving scenarios and are a vital tool in modern cryptography. However the security of OPRFs protocols is heavily reliant on the strength of the keys generated by the server. Due to this dependency, addressing the challenges posed by malicious servers and weakly generated keys is an open problem that necessitates careful consideration.

A key aspect of an OPRF's security is its pseudorandomness, the property that ensures the output of the OPRF is indistinguishable from random. When a weak key is used in an OPRF protocol, the pseudorandomness of the function may be compromised since this can lead to the OPRF using a key that has insufficient entropy, possibly making it easier for adversaries to distinguish the OPRF output from a truly random function. In particular, the pseudorandom function becomes vulnerable to attacks such as brute-force guessing or cryptanalytic attacks, where an attacker can attempt to reverse-engineer or predict the key. This lost of pseudorandomness may have several serious implications for OPRF protocols since OPRFs protocols are used in privacy-preserving applications a lost of pseudorandomness can possibly allow adversaries to infer the inputs being processed, violating the privacy of the users involved in the protocol.

In some OPRFs constructions, VOPRFs (Verifiable Oblivious Pseundoramdom Functions), the client can ensure that the server is behaving honestly by verifying proofs using the public key generated by the server. So if the server chooses (intentionally or not) a non-uniform key, then from the perspective of the client all checks pass, and the client might now, for example, encrypt his data with a weak key. This is a big problem for OPRFs and their implementation and its important to work on integrating well-formedness of keys into OPRFs. This has not yet been considered, but would significantly improve security guarantees of many applications.

It is also interesting to study if the current approach of placing client trust in the server-based key rotation process can lead to concrete cryptographic vulnerabilities in existing systems. This means that there might be cases where if a server has been corrupted and can no longer derive truly random keypairs, there may exist potential scenarios in which even rotating the server key no longer provides appropriate security guarantees. It is also interesting and valuable to investigate the possibility of devising an alternative where the clients are able to derive stronger trust guarantees from this type of protocols instead of placing their full trust on the server's key rotation and key generation.

1.2.3 Impact of the OPRF Key in OPRF Protocols and Applications

It is also interesting to look at applications and protocols that use OPRFs and study the impact that the key has on the security of those protocols.

One good example of a type of protocol that uses OPRFs is the OPAQUE protocol [21], a password-based authentication protocol that provides secure and privacy-preserving authentication for users in a client-server setting. OPAQUE's primary objective is to

establish a secure and confidential channel between a client and a server without directly revealing the user's password to the server. Instead, it utilizes OPRFs to perform secure computations on the password, ensuring that the server gains no knowledge of the password itself. By doing so, OPAQUE effectively prevents the exposure of sensitive user information even in the event of a server compromise or a malicious insider attack. This way OPAQUE not only provides strong security guarantees but also offers resilience against server-side attacks, such as offline dictionary attacks and password database breaches. However, despite the robust security guarantees offered by OPAQUE, the protocol's effectiveness heavily relies on the proper generation and management of OPRF keys. If the server chooses a weak or compromised key, it can potentially compromise the security guarantees of the OPAQUE protocol. This highlights the importance of key rotation security and the need for thorough analysis of the impact of weak keys on the overall security of protocols utilizing OPRFs.

This is a open problem regarding OPRFs protocols, investigating the impact of weak keys in protocols like OPAQUE and analyzing what are the specific security guarantees changes in that case can help to a broader understanding of security protocols that incorporate OPRFs and help their standardization. It is also interesting to investigate if there are ways to mitigate these security losses.

1.3 Expected Contributions

The main goal of this thesis will be to study the impact that keys have in OPRF security and in OPRF protocols. We want to investigate what can happen when the server samples(intentionally or not) a non-uniform key and if that can lead to concrete cryptographic vulnerabilities in existing systems. More specifically, if a server has been corrupted and can no longer derive truly random keypairs, we will identify potential scenarios in which even rotating the server key no longer provides appropriate security guarantees.

We will then investigate the possibility of devising an alternative where we can have a more transparent key rotation strategy that allow clients to derive stronger trust guarantees from the entire exchange by changing the current approach of solely relying on the key generated by the server into a solution that would allow us to argue security over both the input and key distributions, with the idea of being able to derive randomness for the OPRF not only from the server key but also from the client's input.

With this in mind we will aim to answer the following research questions:

- **Question 1**: If a server has been corrupted and can no longer derive truly random keypairs how does that affect the security guarantees of this type of protocols?
- **Question 2**: Is it possible to create an alternative to current OPRFs protocols that allow clients to derive stronger trust guarantees from the entire exchange?

Considering these goals the expected contributions for this thesis are:

- Investigate the impact of keys sampled from non-uniform distributions in the security of OPRF protocols.
- Contribute to a broader understanding of security protocols that incorporate OPRFs.
- Provide insights into the implications of sampled weak keys in OPRFs protocols and applications and explore potential strategies for mitigating their impact.

In conclusion we delve into the study of Oblivious Pseudorandom Functions and in their applications. Specifically, we focus on investigating the vulnerabilities and potential security implications that arise from the server's choice of weak keys, both unilaterally and non-unilaterally. By examining these scenarios, we aim to improve our understanding of the criticality of secure key generation and management and the potential mitigation strategies required to maintain the desired security properties in privacy-preserving cryptographic protocols and applications that use OPRFs.

1.4 Document Organization

The document is organized as follows:

- **Chapter 1** - The first chapter introduces the motivation and objectives of this thesis.
- **Chapter 2** - The second chapter details important concepts and some cryptographic background to better understand the document work.
- **Chapter 3** - In the third chapter we introduce our PRF definition and our security model.
- **Chapter 4** - In the fourth chapter we study a specific OPRF construction and its underlying PRF with the objective of investigating the impact on security of a non-uniform key being used.
- **Chapter 5** - The fifth chapter explores the possibility of mitigating the loss of security analyzed in the fourth chapter by using the concept of a Dual PRF and exploring the possibility of deriving randomness from the client input to avoid relying only on the server key.
- **Chapter 6** - In the sixth chapter we discuss the general implications for deployed protocols based on the results of the investigation in this thesis.
- **Chapter 7** - The last chapter concludes the dissertation, glancing over the main contributions and possible future work.

2

Background and Related Work

In this chapter, we go through and analyze some of the most important concepts related to the thesis work. This will help establish the cryptographic notation and concepts that we will be using in this thesis.

2.1 Modern Cryptography Principles and Theory

First of all it is important to note that in this thesis, we have followed the framework and formalization presented in Katz and Lindell's *Introduction to Modern Cryptography* (2nd edition) [74]. Their comprehensive approach heavily informs the cryptographic definitions, principles, and notation used throughout this thesis. By adopting this framework, we aim to ensure consistency with widely accepted cryptographic practices and terminologies as outlined in their book.

In this section, we will begin by discussing the foundational principles of modern cryptography, including provable security and its relationship to real-world security. Following this, we will introduce key cryptographic concepts that are essential for the analysis in this thesis, such as polynomial-time algorithms, negligible functions, and other fundamental notions used in cryptographic proofs.

2.1.1 Principles of Modern Cryptography

Modern cryptography is grounded in a rigorous and formal approach to analyzing the security of cryptographic protocols. The goal is to move beyond intuitive security guarantees and instead establish formal proofs of security that can withstand adversarial scrutiny. This approach is built on three key principles that we mentioned in the introduction:

- **Formal Definitions** provide a clear and unambiguous specification of the security goals that cryptographic schemes aim to achieve. Without these definitions, it would be impossible to rigorously assess the security of a protocol.

- **Precise Assumptions** establish the computational difficulty of specific problems that underpin the security of cryptographic schemes. They define the adversarial models within which we evaluate the resilience of cryptographic protocols.
- **Proofs of Security**, which often rely on reduction arguments, demonstrate that breaking a cryptographic scheme is as difficult as solving a well-established mathematical problem. These proofs offer a guarantee that a scheme remains secure unless the underlying assumptions are violated.

These proofs are critical in modern cryptography because they provide rigorous guarantees about the security of a protocol, as opposed to relying on heuristic or empirical evidence. A cryptographic scheme is considered provably secure if it can be mathematically demonstrated that no adversary can break the scheme unless they solve an underlying hard problem, such as factoring large integers or solving discrete logarithms.

However, while provable security provides strong guarantees within an idealized model, it does not always fully capture the complexities of real-world deployments. A proof of security is always relative to a specific definition and the assumptions made. If these assumptions or definitions do not accurately model the real-world scenario in which the protocol is deployed, then the security guarantee may not hold in practice.

Additionally, real-world security encompasses factors that are not always captured in formal proofs. Implementation errors, side-channel attacks, and hardware vulnerabilities are some examples of real-world issues that may compromise a provably secure scheme. Even when a scheme is proven secure under certain assumptions, real-world systems must be carefully designed and tested to ensure that these assumptions hold in practice.

So it is important then to understand that provable security of a scheme does not necessarily imply security of that scheme in the real world. Provable security needs cryptographers to continually refine and investigate their definitions and assumptions to more closely match and adapt to the real world, this way provable security does not eliminate the ongoing struggle between attackers and defenders, but it establishes a framework that helps tip the balance in favor of the defender. A good example of such a framework that plays an important role in bridging the gap between provable and real-world security is the Universal Composability (UC) framework [30]. The UC framework allows for modular security proofs, ensuring that a cryptographic protocol remains secure even when composed with other protocols. This is crucial for real-world applications where different cryptographic protocols are often combined in complex ways. The UC framework provides a strong foundation for ensuring that provably secure schemes can be deployed securely in practice .

2.1.2 Security Parameter and Negligible Functions

In modern cryptography, the concept of security parameter and negligible functions is fundamental to the formulation and analysis of security guarantees. In cryptography we

use the concept of an integer-valued security parameter (denoted by λ) that parameterizes both cryptographic schemes as well as all involved parties. When an honest party initializes a scheme (e.g, when it generates a key), it chooses some value λ for the security parameter, this security parameter is assumed to be known to any adversary and we can look at it more intuitively by thinking of the security parameter as the length of the key. In cryptography, this allows us to model the running time of the adversary and its success probability as functions of the security parameter, rather than as concrete numbers.

Now that we have explained the concept of the security parameter, we will move on to the definition of negligible functions, explaining their importance, and illustrating their application in cryptographic proofs.

2.1.2.1 Definition

A function $\text{neg} : \mathbb{N} \to \mathbb{R}^+$ is said to be *negligible* if for every positive polynomial $p(\lambda)$, there exists an integer λ_0 such that for all $\lambda > \lambda_0$, the following inequality holds:

$$\text{neg}(\lambda) < \frac{1}{p(\lambda)}.$$

Formally, we write:

$$\text{neg}(\lambda) = O\left(\frac{1}{p(\lambda)}\right).$$

This definition implies that a negligible function decreases faster than the reciprocal of any polynomial as the security parameter λ grows large. Therefore, as λ increases, $\text{neg}(\lambda)$ approaches zero more rapidly than any inverse polynomial.

2.1.2.2 Intuition

In the context of cryptography, negligible functions are used to quantify probabilities or advantages that are so small that they can be considered effectively zero in practice for sufficiently large security parameters λ. For instance, when we assert that the probability of an adversary successfully breaking a cryptographic scheme is negligible in λ, we mean that the adversary's success probability decreases so rapidly as the security parameter increases that it becomes impractical for any adversary to achieve a meaningful advantage.

Negligible functions thus allow us to formalize the notion of computational infeasibility in a rigorous manner, which is essential for providing concrete security guarantees in cryptographic protocols and primitives.

A common example of a negligible functions is $2^{-\lambda}$, which represents an exponentially decreasing function. This function illustrates the idea that, as λ grows, the function value diminishes to a point where it becomes insignificant.

2.1.2.3 Application in Cryptographic Proofs

Negligible functions are crucial in cryptographic security proofs. For example, consider a cryptographic protocol where the success probability of an adversary $\mathcal{A}$ in breaking the protocol is denoted by $\Pr[\text{Adv}_{\mathcal{A}}(\lambda)]$. If we can show that this probability is bounded by a negligible function $\text{neg}(\lambda)$, we can conclude that the protocol is secure against $\mathcal{A}$ for sufficiently large λ.

In such proofs, the notion of negligible probability allows cryptographers to make strong statements about the security of a scheme. Specifically, the statement "the probability of breaking the scheme is negligible" indicates that while the adversary's success might not be zero, it is so small that it is negligible for all practical purposes.

This rigorous approach enables cryptographers to provide formal and precise security assurances that are essential in both theoretical and applied cryptography.

2.1.3 Probabilistic Polynomial-Time (PPT) Algorithms

In the world of cryptography we rely on algorithms capable of making probabilistic choices within polynomial time, known as Probabilistic Polynomial-Time (PPT) algorithms. As Katz and Lindell [74] explain, this notion is fundamental in the formulation of secure cryptographic definitions and security models since PPT algorithms are ingrained in the theoretical principles and frameworks of how cryptographic systems are designed, analyzed, and understood.

The notion of PPT algorithms is useful for various reasons in cryptography, including efficient and realistic modeling, adaptive adversarial modeling, cryptographic assumptions of hardness notions, and attack feasibility.

2.1.3.1 Efficiency and Realistic Modeling

Cryptographic systems operate in the real world where computational efficiency is a critical consideration that we need to have, because of this PPT algorithms are useful in cryptography since they provide a realistic model of computation.

PPT algorithms are designed to run in polynomial time, which means that their computation time is proportional to a polynomial function of the input size. In mathematical terms this means that if an algorithm runs in polynomial time, the time complexity $T(\lambda)$ is upper-bounded by a polynomial function $P(\lambda)$, where λ is the size of the input. Mathematically, this can be expressed as:

$$T(\lambda) = O(P(\lambda))$$

This is why, for example, that an algorithm that has a time complexity of $O(\lambda^2)$ is a polynomial time algorithm because the running time is proportional to the square of the input size. In contrast to this, algorithms that run in exponential time, factorial time, or other non-polynomial time complexities are generally considered less efficient, especially

for large inputs. Polynomial time algorithms are generally more practical and scalable, and they are often desirable in practical applications due to their efficient nature.

Because of this, as Katz and Lindell [74] refer, in modern cryptography we are only interested in adversaries whose running time is polynomial in the security parameter λ. Since we measure the running time of an algorithm in terms of the length of its input, we sometimes provide algorithms with the security parameter written in unary (i.e., as 1^λ, or a string of n ones) as input. Parties (or, more precisely, the algorithms they run) may take other inputs besides the security parameter, for example, a message to be encrypted, and we allow their running time to be polynomial in the (total) length of their inputs.

Cryptographic systems need to operate within practical constraints, considering factors like limited computational power. Because of their polynomial-time nature, the notion of PPT algorithms is useful in cryptography for realistic modeling, ensuring that cryptographic definitions and security models align with the constraints of real-world computing environments.

2.1.4 Adaptive Adversarial Modeling

Another key reason for using PPT algorithms in cryptographic definitions is the need to realistically model adaptive adversaries. Unlike deterministic algorithms, PPT algorithms can make probabilistic choices during their execution, providing a more accurate representation of adversaries in real-world scenarios. This adaptive modeling is crucial in assessing the security of cryptographic primitives and protocols against sophisticated and dynamic threats.

2.1.5 Cryptographic Assumptions and Hardness Notions

Many cryptographic proofs, security models and security reductions rely on the assumption that certain computational problems are difficult to solve in probabilistic polynomial time. PPT algorithms serve as the foundation for defining computational hardness notions, forming the basis for a lot of cryptographic assumptions. For instace, the hardness of problems like factoring large integers or computing discrete logarithms is often expressed in terms of PPT algorithms.

2.1.6 Feasibility of Attacks

In the analysis of cryptographic systems, it is essential to evaluate the feasibility of potential attacks. PPT algorithms provide a yardstick for assessing computational feasibility. If an adversary could break a cryptographic scheme in polynomial time, it would signify a significant vulnerability. The use of PPT algorithms allows researchers to gauge the practicality of attacks and design resilient cryptographic solutions.

2.2 Keys in Cryptography

In modern life, cryptographic keys play an essential role in ensuring the security and privacy of digital communication, financial transactions, and sensitive data storage. Whether unlocking a smartphone, securely accessing an online banking account, or protecting confidential messages in messaging apps, cryptographic keys serve as the foundation for authenticating users and encrypting information.

In practice, a cryptographic key is a value that algorithms use to transform plaintext into ciphertext in encryption schemes or vice versa in decryption schemes. In private key encryption (symmetric cryptography), the same key is used for both encryption and decryption. An example of such a system is AES (Advanced Encryption Standard) [40]. In these cases, the key must remain secret between communicating parties because if an adversary obtains the key, they can decrypt the ciphertext and access sensitive information.

Keys also play an essential role in public-key cryptography, where two distinct, but mathematically related, keys are used: a public key (that can be shared openly) and a private key (that is kept secret). The public key is often used for tasks like encrypting data or verifying digital signatures, while the private key is used for decrypting data that was encrypted with the corresponding public key or for creating digital signatures that prove authenticity. This system allows two parties to communicate securely, even if they have never met before, because the public key can be shared openly, and only the holder of the private key can decrypt or authenticate messages related to that key.

Protocols like RSA [102] and ECC [76] rely on these key pairs, where security hinges on the difficulty of solving problems like integer factorization or the discrete logarithm problem. The strength of these systems depends not only on the key length but also on how the keys are generated, stored, and managed throughout their lifecycle.

Because of this, protocols and primitives that use cryptographic keys fundamentally depends on the strength and proper management of the keys.

2.2.1 Key Strength

Typically, when we talk about key strength, we think about key length. Key length refers to the number of bits in the key, which directly influences the size of the key space. For a key of length λ bits, the total number of possible keys is 2^{λ}. For example, a 128 bit key provides 2^{128} possible key combinations. The larger the key space, the more computationally expensive it is for an adversary to search through all possible keys in a brute-force attack.

However, the effective key strength is not only a matter of key size. It also depends on the randomness of the key generation process and the distribution from which the key is drawn. Poorly generated or non-uniform keys can drastically reduce the security of a system, as attackers could exploit patterns in the key generation process to narrow the search space.

A weak key, in traditional cryptographic settings, is one that does not provide sufficient

security, either because it is too short or due to issues related to how it is generated or managed. Several factors can result in a weak key such as short key lengths, low entropy and structural weaknesses in the key distribution.

When it comes to OPRFs, key strength is an interesting research topic, as weak keys may impact the pseudorandomness required for security.

2.2.2 Key Rotation and Key Generation

Key rotation is an essential aspect of key management. It involves periodically replacing cryptographic keys with new ones to reduce the potential impact of key compromise. If a key is compromised, rotating it ensures that any future communications or data are protected by a fresh, uncompromised key. This approach is particularly crucial in long-term communication systems, where the same key is used across multiple sessions or extended periods.

Rotating keys helps mitigate risks related to key exhaustion, where a cryptographic key's use over time could provide an attacker with enough material to attempt a brute-force or cryptanalytic attack. In systems like TLS [24, 53], key rotation mechanisms are implemented to ensure that even if an attacker compromises one session key, the exposure is limited to that session, as future sessions will use new keys.

Key rotation implies generating new keys, key generation is a critical process that produces the cryptographic keys used in encryption, decryption, signing, and verification. For a key to be secure it must be unpredictable to attackers, which is why randomness is essential for generating secure cryptographic keys. There are two main types of randomness used in key generation:

- **True Randomness:** This is derived from unpredictable physical processes such as radioactive decay, thermal noise, or atmospheric noise. True random number generators (TRNGs) use these unpredictable sources to produce random bits. Since true randomness comes from natural phenomena, it is highly unpredictable and provides high entropy, making it ideal for key generation. However, accessing sufficient true random data can be challenging and slow in certain environments, which limits its use in many real-time cryptographic applications, nevertheless there are some cases of its use like for example Cloudfare that uses lava lamps as a natural source of randomness to help with encryption [38].

- **Pseudorandomness:** Due to the limitations of gathering true randomness, many cryptographic systems rely on pseudorandom number generators (PRNGs). These generators use deterministic algorithms that, given an initial seed (which should be a truly random value), produce sequences of numbers that are computationally indistinguishable from true random numbers. A special type of PRNG, called a cryptographically secure pseudorandom number generator (CSPRNG), is typically

used in key generation because its output is designed to be unpredictable even if part of the seed or the output sequence is known.

2.2.3 The Challenge of Realizing Cryptographic Tasks Without Uniform Randomness

In the field of cryptography, significant effort has been dedicated to studying whether cryptographic tasks can be performed without perfect access to uniform randomness. Traditional cryptographic protocols assume the availability of a perfectly random source to generate keys, which is not always feasible in real world scenarios.

Several studies [2, 20, 49, 50, 51, 52, 85] have addressed this challenge by investigating whether it is possible to achieve cryptographic tasks using non-uniform or imperfect randomness. For example, the work of Dodis et al. [51] explores the feasibility of executing cryptographic tasks with entropy sources that provide less-than-perfect randomness. Their findings indicate that while some cryptographic tasks, like encryption and zero-knowledge proofs, are impossible under such conditions, other tasks, like secure signature schemes, can still be achieved under certain assumptions.

Some of this research on randomness has demonstrated that the assumption of uniform randomness can be relaxed in certain settings, but this often comes with trade-offs in security. Techniques like fuzzy extractors [31], bounded leakage models [4], and secure key derivation functions [79] can sometimes ensure security despite the presence of weak keys, as long as certain conditions are met, such as the availability of additional randomness or structure in the protocol. Entropy-based approaches to randomness generation, also offer some solutions by guaranteeing high entropy while acknowledging that perfect randomness might not always be attainable.

2.2.4 Entropy and Its Role in Cryptographic Security

Because of this challenge, the concept of entropy becomes very relevant in cryptography. Entropy measures the unpredictability or randomness of a cryptographic key, which can directly influence the strength of the cryptographic protocol. A key that has low entropy (meaning that it lacks sufficient randomness), can be guessed by an attacker, rendering encryption ineffective. Therefore, understanding and applying entropy, particularly min-entropy, is crucial in the design of secure cryptographic systems.

Cryptographic systems rely on randomness to ensure security. Randomness is used in various contexts, including key generation, nonce selection, and the construction of cryptographic primitives such as PRFs and encryption schemes. To quantify the quality of randomness, cryptographers use the concept of entropy, which measures the unpredictability of a random variable. The more unpredictable the random variable, the higher the entropy. Two specific types of entropy that are particularly relevant in cryptography are Shannon entropy and min-entropy.

2.2.4.1 Shannon Entropy

Shannon entropy, introduced by Claude Shannon in 1948 [105], is the most commonly known measure of unpredictability and uncertainty in information theory. It quantifies the average amount of information produced by a random variable. Let X be a discrete random variable with possible outcomes $x_1, x_2, \ldots, x_n$, each occurring with probability $p(x_i)$. The Shannon entropy $H(X)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i).$$

Shannon entropy measures the expected amount of uncertainty in a random variable. If all outcomes are equally likely, the entropy is maximized. For example, if X is a uniformly distributed random variable over n possible outcomes, the entropy is $logn$. However, Shannon entropy is an average measure, which means that it may not adequately capture worst-case scenarios, where some outcomes are much more likely than others. This is why other forms of entropy, such as min-entropy, are used in cryptography.

2.2.4.2 Min-Entropy

In cryptography, min-entropy [110] is often more relevant than Shannon entropy, as it focuses on the worst-case predictability of a random variable rather than its average behavior. Min-entropy is the notion that is most widely used for modeling weak sources of randomness [34]. This model was introduced by Chor and Goldreich [37] and Zuckerman [112].

Min-entropy measures the uncertainty associated with the most likely outcome of a random variable. Formally, for a random variable X, the min-entropy $H_\infty(X)$ is defined as:

$$H_\infty(X) = -\log \max_x \Pr[X = x]$$

If the min-entropy of a random variable X is low, it means that there is a high likelihood that X takes on a particular value with significant probability. In contrast, if the min-entropy is high, the random variable is well-distributed and hard to predict.

2.2.4.3 Relevance of Min-Entropy in Cryptography

Min-entropy measures the worst-case unpredictability by quantifying the probability that an adversary can correctly guess the secret in a single attempt. Unlike Shannon entropy, which averages over all possible outcomes, min-entropy focuses on the likelihood of the most probable event. This makes it a more conservative and thus more appropriate measure in cryptographic contexts, where we are concerned with the success probability of an adversary who may have some prior knowledge about the secret.

In cryptography, what is of primary interest is setting tight bounds on the adversary's success probability in any attack. The goal is to ensure that even in the worst-case scenario, the adversary's chances of guessing a secret value (such as a key, nonce, or password) remain sufficiently low. Min-entropy is directly connected to this concept because it defines the maximum probability with which the adversary could predict the value. Higher min-entropy implies lower success probability for the adversary, translating to stronger security guarantees.

Min-entropy is particularly important in cryptographic applications where even the slightest bias in a random variable can lead to security vulnerabilities. Cryptographic keys, for example, should be generated from distributions with high min-entropy to reduce the probability that an adversary can predict them with significant likelihood. If the key distribution has low min-entropy, it means that certain keys are much more likely than others, making it easier for an attacker to guess or brute-force the key. In such cases, adversaries can conduct distinguishing attacks, where they interact with the cryptographic protocol multiple times and, due to the low min-entropy of the key, can detect patterns or predict the secret key with non-negligible probability.

In modern cryptography, the use of high-min-entropy sources is a fundamental requirement for ensuring the security of cryptographic protocols. Entropy sources are typically tested for min-entropy to verify that they provide sufficient unpredictability, and cryptographic standards (e.g., NIST) emphasize the need for high-min-entropy key generation processes [106]. As such, min-entropy is often used to set formal security thresholds in cryptographic protocols, ensuring that secrets used in encryption, key exchange, and authentication processes meet the security requirements.

2.3 Cryptographic Primitives

Cryptographic primitives are the foundational building blocks used to design secure cryptographic protocols and systems. These primitives are typically simple, well-defined mathematical functions or algorithms that exhibit specific security properties. Cryptographic primitives form the core components of more complex protocols. This section provides an overview of the most commonly used cryptographic primitives, which are essential for understanding the security guarantees and constructions in modern cryptographic systems.

- **Symmetric-Key Primitives**: These rely on a shared secret key between parties. The most common examples are block ciphers such as AES [40], which operate on fixed-size blocks of data, and stream ciphers like RC4 [103], which encrypt data in a continuous stream.

- **Public-Key Primitives**: Public-key cryptography uses a key pair (public and private). Prominent examples include the RSA [102] encryption scheme, based on the difficulty

of factoring large integers, and Elliptic Curve Cryptography (ECC) [76], which offers similar security with smaller key sizes.

- **Cryptographic Hash Functions**: Hash functions, such as SHA-256 [107], map arbitrary data to fixed-size outputs. They are widely used for ensuring data integrity and in applications like digital signatures and HMACs.

- **Pseudorandom Functions (PRFs) and Pseudorandom Generators (PRGs)**: PRFs produce pseudorandom outputs from a secret key and a message input, and PRGs expand a short random seed into a longer pseudorandom sequence, both essential for encryption and key generation.

- **Message Authentication Codes (MACs)**: MACs are used to verify the integrity and authenticity of a message. A widely used MAC scheme is HMAC [80], which combines a hash function with a secret key.

We will now delve into more detail on specific primitives that are particularly relevant for the analysis in this thesis, including Pseudorandom Generators, Pseudorandom Functions, and Dual PRFs.

2.3.1 Pseudorandom Generators (PRGs)

A PRG is a deterministic algorithm that expands a short, truly random seed into a longer sequence of bits that appears random to any efficient adversary. Formally, given a random seed of length λ, a PRG produces an output of length $\ell(\lambda)$, where $\ell(\lambda) > \lambda$. The key property of a PRG is that its output is computationally indistinguishable from a truly random sequence of length $\ell(\lambda)$.

Definition 1 (PRG [74])**.** *Let ℓ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any λ and any output $s \in \{0,1\}^\lambda$, the result $G(s)$ is a string of length $\ell(\lambda)$. We call ℓ the expansion factor of G. We say that G is a PRG if the following conditions hold:*

1. ***Expansion:*** *For every λ it holds that $\ell(\lambda) > \lambda$.*

2. ***Pseudorandomness:*** *For any PPT adversarie $\mathcal{A}$, there is a negligible function negl such that*

$$|\Pr[\mathcal{A}(G(s)) = 1] - \Pr[\mathcal{A}(r) = 1]|$$

where the first probability is taken over the uniform choice of $s \in \{0,1\}^\lambda$ and the randomness of $\mathcal{A}$, and the second probability is taken over the uniform choice of $r \in \{0,1\}^{\ell(\lambda)}$ and the randomness of $\mathcal{A}$.

This notion is fundamental for applications requiring a large amount of pseudorandom data derived from a small, high-entropy seed, ensuring the expanded output appears random and unpredictable. It is important to note that while PRGs do stretch small amounts of data, the quality of the seed matters. If the seed doesn't have sufficient entropy, the pseudorandomness produced by the PRG may not be secure. Therefore, good entropy sources are essential for the seed itself, even though PRGs generate large amounts of data from that small seed . .

PRGs play a crucial role in various cryptographic constructions, including stream ciphers, where a pseudorandom keystream is generated from a short secret key, and key generation protocols, where secure keys are derived from small random seeds [59, 74, 93]. In the context of this thesis, PRGs are particularly relevant in understanding how randomness can be derived and expanded efficiently, a concept that underpins many cryptographic systems.

2.3.1.1 Key Generation

PRGs play a vital role in cryptographic key generation. Secure keys are essential for the robustness of cryptographic protocols, and PRGs provide a means to derive pseudorandom keys from shorter random seeds.

2.3.1.2 Efficient Use of Resources

PRGs enable the efficient use of random bits. Cryptographic systems often operate in resource-constrained environments, and the ability to expand a limited amount of true randomness into a larger pseudorandom sequence is invaluable.

In the context of OPRFs and the significance of secure key sampling, PRGs become integral. The pseudorandom keys generated by PRGs contribute to the unpredictability and privacy of the OPRF protocol. The reliance on a robust PRG ensures that the pseudorandom keys used in OPRFs exhibit the necessary cryptographic properties to withstand potential attacks.

2.3.2 Pseudorandom Functions (PRFs)

Pseudorandom functions have been introduced by Goldreich, Goldwasser and Micali [60], they are cryptographic primitives, keyed functions that behave like random functions but are actually deterministic and computationally efficient. A PRF takes an input key and an input message and produces an output that appears random, even to a computationally powerful adversary. PRFs have broad utility in cryptographic constructions such as, being used in secure comunication protocols like TLS [47], being a building block for OPRFs [75], being used for key derivation functions (KDFs) [79] and more.

Basically, a PRF is defined as a function F that maps an input key k and an input message x to an output y, where k and x can be bit strings of arbitrary length. Its denotes as follows:

$$y = F(k, x).$$

Below we give give a formal definition of a PRF made by Casacuberta, Hesse, Lehmann [32]:

Definition 2 (Pseundorandom function)**.** *A family of functions $f_k : \{0,1\}^m \to \{0,1\}^n$ with key $k \in \{0,1\}^\lambda$ is called pseudorandom if the following holds:*

- *$f_k(x)$ is efficiently computable from k and x. Meaning that there exists a deterministic algorithm that, given a key $k \in \{0,1\}^\lambda$ and an input $x \in \{0,1\}^m$, computes $f_k(x)$ in polynomial time with respect to m, n, and λ.*
- *It is not efficiently decidable whether one has access to a computation oracle for $F_s(\cdot)$ or to an oracle producing random bitstrings of length n.*

The security of a PRF lies in its ability to resist various attacks, such as distinguishing it from a truly random function. A PRF should exhibit the following main properties:

- **Pseudorandomness:** For any efficient algorithm, it should be computationally infeasible to distinguish the output of the PRF from the output of a random function, given access to multiple queries of the function with different inputs. Formally, we say that for any probabilistic polynomial-time adversary $\mathcal{A}$ with oracle access, the adversary's advantage in distinguishing between the following two scenarios is negligible in λ:
  - The oracle is a function f_k, where k is chosen uniformly at random from $\{0,1\}^\lambda$.
  - The oracle is a truly random function that outputs uniformly random bitstrings of length n for each distinct input x.

  Formally, for any probabilistic polynomial-time adversary $\mathcal{A}$, the distinguishing advantage is given by:

  $$\left|\Pr[\mathcal{A}^{f_k(\cdot)} = 1] - \Pr[\mathcal{A}^{R(\cdot)} = 1]\right| \leq \text{negl}(\lambda),$$

  where $R(\cdot)$ denotes a truly random function, and $\text{negl}(\lambda)$ is a negligible function in λ.
- **Keyed Security:** The security of the PRF should rely on the secrecy of the key. Like Katz and Lindell [74] explain, when they introduce the notion of a PRF, that it is meaningless to require that F_k be pseudorandom if k is known, since then it is trivial to distinguish an oracle for F_k from an oracle for f_n given k: simply query the oracle at the point 0^n to obtain the answer y, and compare this to the result $y' = F_k(0^n)$ that can be computed using the known value k. An oracle for F_k will always return $y = y'$, while an oracle for a random function will have $y = y'$ with probability only 2^{-n}.

In practice, this means that once k is revealed, all claims to the pseudorandomness of F_k no longer hold. To take a concrete example: say F is pseudorandom. Then given oracle access to F_k (for random k), it will be hard to find an input x for which $F_k(x) = 0^n$ (since it would be hard to find such an input for a truly random function f_n). But if k is known, then finding such an input may be easy.

- **Efficiency:** A PRF should be computationally efficient, meaning that it should be able to process input messages and produce outputs in a reasonable amount of time. Formaly, there exists a deterministic algorithm that, given a key $k \in \{0,1\}^\lambda$ and an input $x \in \{0,1\}^m$, computes $f_k(x)$ in polynomial time with respect to m, n, and λ. That is, $f_k(x)$ is efficiently computable for all k and x.

- **Resistance to Known Attacks:** A PRF's design and construction should be resilient against known cryptographic attacks, such as differential and linear cryptanalysis, birthday attacks, and related-key attacks.

We will go into greater detail into the security definition of a PRF in Chapter 3.

2.3.3 Dual Pseudorandom Functions (Dual PRFs)

A Dual Pseudorandom Function (dual PRF) is a cryptographic primitive that extends the concept of a PRF to support two independent keys, meaning a PRF that is not only keyed conventionally through its key, but also when "swapped" and keyed (unconventionally) through its input(or message) [13].

Dual PRFs were first introduced in the context of HMAC by Bellare [10, 11]. HMAC (Hash-based Message Authentication Code) [12] is a specific type of message authentication code (MAC) that was designed as a cryptographic-hash function-based PRF taking two inputs, with first input being the key and the second the message. It is widely used, for example in TLS, IPsec and SSH and is standardized by IEFT [80] and NIST [91].

However, more recently it has also been used as a key combiner, which additionaly assumes to function as a swap-PRF (an assumption that has been validated in [13], meaning that it also behaves as a PRF when the second input is treated as the key and the first input as the message, having a dual PRF usage. This dual PRF assumption allowed for strong security proofs of these constructions and is used in several Internet security protocols such as TLS 1.3 [24, 53], hybrid key-exchange [15, 108], KEMTLS [104], post-quantum versions of WireGuard [65] and Noise [5], and Message Layer Security (MLS) [23]. This dual PRF assumption has been specifically analyzed in articles like [6, 13]

Intuitively, a dual PRF is a PRF that remains a PRF when the roles of its input and key are switched. More formally, Backendal et al. [6] presented the following formal definition of a dual-PRF by defining a PRF and a swap-PRF:

The definition of a PRF being a function family $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $K \in \mathcal{K}$ is the key and $X \in \mathcal{X}$ is the input(or message), requires that the oracles for $F(K, \cdot)$ and a random

function $f : \mathcal{X} \to \mathcal{Y}$ be indistinguishable when $K \xleftarrow{\$} \mathcal{K}$ is not known to the attacker [60]. The *swap* of F is the function family $F^{\leftrightarrow} : \mathcal{X} \times \mathcal{K} \to \mathcal{Y}$ defined by $F^{\leftrightarrow}(X, K) = F(K, X)$. We say that F is a swap-PRF if $F^{\leftrightarrow}$ is a PRF. (That is, F is a PRF when keyed by the second, or message, input.)

Definition 3 (Dual-PRF)**.** *Formally, F is a dual-PRF if F is both a PRF and a swap-PRF. That is, it is a PRF when keyed as usual by the first input, but also if keyed by the second input.*

2.4 Cryptographic Protocols

Cryptographic protocols are the building blocks of secure communication in the digital world, they consist on sets of rules and procedures that govern secure communication or computation between multiple parties with the objective of achieving various security goals such as confidentiality, integrity, authentication, and non-repudiation. They are vital in ensuring the security and privacy of modern communication systems [56]. The growing reliance on the internet for sensitive transactions, such as financial exchanges, medical data sharing, and private communication, has made these protocols indispensable. Without strong cryptographic protocols, the internet and digital systems would be vulnerable to attacks like eavesdropping, data tampering, and impersonation.

An example of one of the most well-known cryptographic protocols is the Secure Socket Layer (SSL) [58] or its successor, the Transport Layer Security (TLS) protocol [47, 101] used for securing internet communications by enabling two parties to identify and authenticate each other and to communicate with confidentiality and data integrity. The main goal of both protocols is to provide privacy, data integrity, identification, and authentication. [39]

An example of a very relevant cryptographic protocol in the field of cryptography is the Diffie-Hellman key exchange. This protocol is also specifically relevant for this thesis since it is connected to the OPRF construction we will talk about in Chapter 4 so it is valuable to talk about this protocol in more detail.

2.4.1 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange, a cryptographic protocol published by Whitfield Diffie and Martin Hellman in 1976 [48], revolutionized the field of cryptography by introducing the concept of public key cryptography. At its core, the Diffie-Hellman protocol allows two parties to establish a shared secret key over an insecure communication channel, without the need for prior communication or shared secrets.

This protocol relies on modular exponentiation as a fundamental mathematical operation to securely compute shared secret keys. To understand this, we must first introduce some essential concepts from group theory and modular arithmetic [16].

2.4.1.1 Modular Exponentiation

Modular exponentiation is the operation of raising a number to an exponent and then taking the remainder when divided by a modulus. Formally, given a base g, an exponent x, and a modulus p, the modular exponentiation is defined as:

$$g^x \mod p$$

where g^x denotes the exponentiation of g by x, and $\mod p$ denotes taking the remainder after division by p.

2.4.1.2 Group Definition

A *group* G is a set of elements combined with a binary operation $\cdot$ (often multiplication or addition) that satisfies the following four properties:

1. **Closure**: For any two elements $a, b \in G$, the result of the operation $a \cdot b$ is also in G.

$$\text{If } a, b \in G, \text{ then } a \cdot b \in G.$$

2. **Associativity**: For any three elements $a, b, c \in G$, the equation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds.

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

3. **Identity Element**: There exists an element $e \in G$ such that for every element $a \in G$, the equation $a \cdot e = e \cdot a = a$ holds.

$$\exists\, e \in G \text{ such that } \forall\, a \in G,\ a \cdot e = e \cdot a = a.$$

4. **Inverse Element**: For each element $a \in G$, there exists an element $b \in G$ such that $a \cdot b = b \cdot a = e$, where e is the identity element.

$$\forall\, a \in G, \exists\, b \in G \text{ such that } a \cdot b = b \cdot a = e.$$

2.4.1.3 Multiplicative Group $\mathbb{Z}_p^*$

In cryptographic contexts, we often work within the multiplicative group of non-zero integers modulo a prime number p. This group, denoted as $\mathbb{Z}_p^*$ (where $*$ means that zero is excluded), is defined as the set of integers $\{1, 2, 3, \ldots, p-1\}$ under multiplication modulo p. The group $\mathbb{Z}_p^*$ satisfies the following properties:

- **Closure**: For any $a, b \in \mathbb{Z}_p^*$, $(a \cdot b) \mod p \in \mathbb{Z}_p^*$.
- **Associativity**: The operation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds for all $a, b, c \in \mathbb{Z}_p^*$.
- **Identity Element**: The element 1 serves as the identity, since for any $a \in \mathbb{Z}_p^*$, $a \cdot 1 \equiv a \mod p$.

- **Inverse Element**: For each $a \in \mathbb{Z}_p^*$, there exists an $a^{-1} \in \mathbb{Z}_p^*$ such that $a \cdot a^{-1} \equiv 1 \mod p$.

In cryptography, there is a preference for cyclic groups of prime order. This prime number p is typically chosen to be large since it has been proven that certain mathematical problems that are connected to the security of cryptographic systems, like the discrete-logarithm problem, become easier if p has (small) prime factors [74, 94].

Cyclic Groups and Generators: Another important property of $\mathbb{Z}_p^*$ is that it is cyclic, meaning there exists an element $g \in \mathbb{Z}_p^*$ (called a generator or primitive root) such that every element of the group can be expressed as a power of g, modulo p. Formally, for every element $a \in \mathbb{Z}_p^*$, there exists an integer k such that:

$$a = g^k \mod p$$

where k ranges from 0 to $p - 2$.

Having introduced this concepts, we now come back to the Diffie-Hellman Key Exchange protocol. In this cryptographic scheme, Alice and Bob seek to establish a shared secret key. To do this, they first agree on a generator g and prime p. Then, they both generate their private key x randomly from a large set of possible values (1 to $p - 1$), we will represent Alice's private key as a and Bob's as b.

Then they both compute their own public keys: Alice computes

$$g^a \mod p$$

and Bob computes

$$g^b \mod p$$

Then Alice and Bob exchange public keys, g^a and g^b and then both parties can compute the shared secret key since:

$$(g^b)^a \mod p = (g^a)^b \mod p$$

Both computations result in the same shared secret key, which both parties can use for secure communication.

To obtain the private keys in this protocol, an adversary would need to extract a and b from the values g^a and g^b, respectively. The problem of determining the exponent x in g^x is known as the Discrete-Logarithm Problem (DLP). As the prime p increases, computing x from g^x becomes increasingly difficult, while computing g^x remains computationally easy. In cryptography, we often rely on the assumption that computing the discrete logarithm is hard, as no efficient general method is known for solving this problem in a cyclic group.

Katz and Lindell [74] formally define the discrete-logarithm problem as follows:

The discrete logarithm experiment $\mathrm{DLog}_{\mathcal{A},\mathcal{G}}(\lambda)$

Consider the following experiment for a group-generation algorithm $\mathcal{G}$, algorithm $\mathcal{A}$, and security parameter λ:

1. Run $\mathcal{G}(1^\lambda)$ to obtain output $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order p (with $|p| = \lambda$) and g is a generator of $\mathbb{G}$.
2. Choose an uniform $h \in \mathbb{G}$.
3. $\mathcal{A}$ is given $\mathbb{G}, p, g, h$, and outputs $x \in \mathbb{Z}_q$.
4. The output of the experiment is defined to be 1 if $g^x = h$, and 0 otherwise.

Definition 4 (Discrete-Logarithm problem)**.** *We say the discrete logarithm problem is hard relative to $\mathcal{G}$ if for all probabilistic, polynomial-time algorithms $\mathcal{A}$ there exists a negligible function $negl(\lambda)$ such that*

$$\Pr[DLog_{\mathcal{A},\mathcal{G}}(\lambda) = 1] \leq negl(\lambda).$$

Besides this assumption there are two other assumptions to help prove the security of this protocol, these assumptions are the Computational Diffie–Hellman (CDH) assumption and the Decisional Diffie–Hellman (DDH) assumption, both detailed in [17], which respectively assume that computing g^{ab} from g^a and g^b is also a hard problem, and that an adversary cannot learn something about g^{ab} or predict the value with some probability since g^{ab} looks exactly like a random element from the group. We formally define these two assumptions below:

Definition 5 (Computational Diffie-Hellman (CDH) assumption)**.** *Let $\mathbb{G}$ be a cyclic group with a generator g. We say that the CDH problem is hard relative to G if for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function $negl(n)$ such that*

$$\Pr\left[\mathcal{A}(g, g^a, g^b) = g^{ab}\right] \leq negl(n),$$

where the probabilities are taken over the random choices of $a, b \in \mathbb{Z}_q$ and the random bits of $\mathcal{A}$.

The security of Diffie-Hellman key exchange relies on this hardness of the Decisional Diffie-Hellman problem, that asserts that it is difficult to distinguish between a valid Diffie-Hellman tuple and a random tuple in a cyclic group of prime order, more precisely the DDH assumption states that no polynomial-time algorithm can distinguish between a Diffie-Hellman tuple (g, g^a, g^b, g^{ab}) and a random tuple (g, g^a, g^b, g^c) with a non-negligible advantage, where $a, b, c \in \mathbb{Z}_q$ are chosen uniformly at random.

Definition 6 (Decisional Diffie–Hellman (DDH) assumption)**.** *Formally, let $\mathbb{G}$ be a cyclic group of prime order q and generator g. The DDH problem is hard in $\mathbb{G}$ if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function $negl(n)$ such that:*

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq negl(n),$$

where the probabilities are taken over the random choices of $a, b, c \in \mathbb{Z}_q$ *and the random bits of* $\mathcal{A}$.

This protocol is one of the most relevant cryptographic protocols ever since it enabled secure communication without the need for pre-shared secrets or a secure initial channel. Besides that it had a profound impact on the field of cryptography, serving as the catalyst for numerous research papers, proposals, and cryptographic systems, laying down the foundations for countless cryptographic applications in today's interconnected world.

2.4.2 Oblivious Pseudorandom Functions (OPRFs)

OPRFs are specific cryptographic protocols designed to enable two parties to securely compute the output of a PRF while maintaining privacy.

In this protocol one of the parties (known as the server) holds the secret key k for a secure pseudorandom function F and the other party (known as the client) has an input x for the function. Through a series of interactions the two parties jointly compute the output of the PRF. At the end of the protocol the client should know $F(k, x)$ and nothing else while the server learns nothing. This way the protocol ensures that the server does not learn the client's input x during evaluation and that the client doesn't learn anything about the the server's secret PRF key [42].

The concept of OPRFs was introduced to address the limitations of traditional pseudorandom functions in scenarios where one party needs to evaluate a function on private inputs held by another party, since then OPRFs have gained significant attention in recent years due to their numerous applications, such as oblivious keyword search (KS) [57, 77], private information retriaval (PIR) [57], password-protected secret sharing (PPSS/TPASS) [67], password-authenticated key exchange (PAKE) [67], private set intersection(PSI) [64, 71, 72, 77] and cloud key management [68].

However OPRFs are still evolving, and there is ongoing work in developing standardized protocols and ensuring interoperability between different implementations. There are various practical aspects of deploying OPRFs in real-world applications and some documents and articles have already brought to attention some open problems and future research directions related to OPRFs [32].

Below we give a formal definition of what is an OPRF:

Definition 7 (Oblivious pseudorandom function, [57])**.** *A two-party protocol* π *between a client and a server is an oblivious pseudorandom function (OPRF) if there exists some PRF family* f_k*, such that* π *privately realizes the following functionality:*

- *Client has input* x*; Server has key* k*.*
- *Client outputs* $f_k(x)$*; Server outputs nothing.*

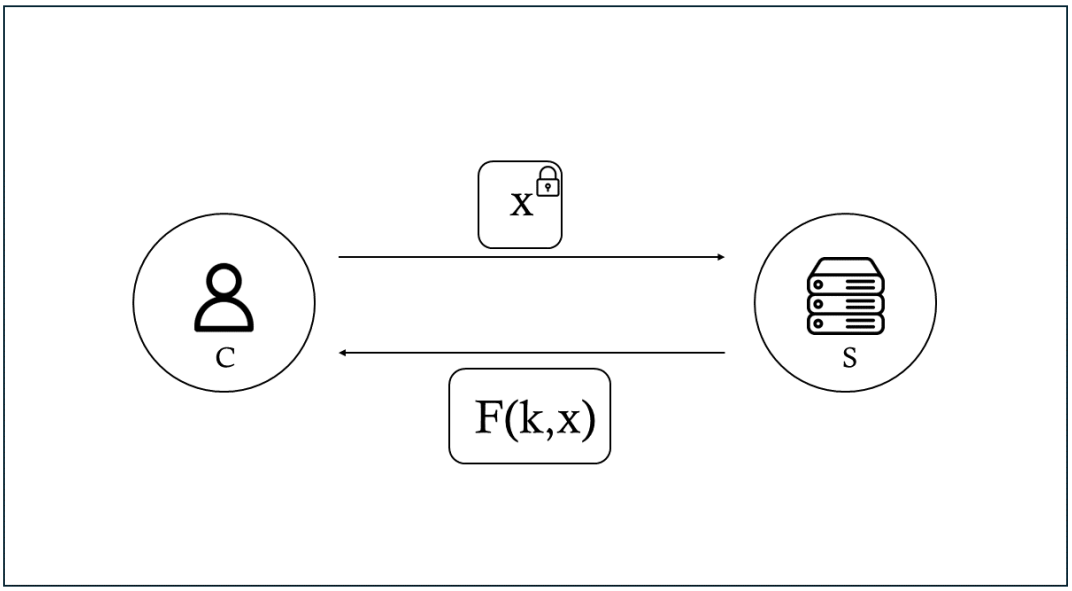

Figure 2.1: General OPRF protocol diagram

In this context, *"π privately realizes"* means that the server learns nothing about the client's input x, and the client only learns the output $f_k(x)$ without obtaining any information about the server's secret key k. That is, while both parties jointly compute the output $f_k(x)$, the server remains oblivious to the input x, and the client remains oblivious to the key k. Formally:

- **Client has input** x: The client wants to compute the value of the pseudorandom function $f_k(x)$, but does not know the secret key k.
- **Server has the key** k: The server knows the secret key k, but does not know the input x chosen by the client.
- **Client outputs** $f_k(x)$; **Server outputs nothing**: At the end of the protocol, the client learns $f_k(x)$ (the result of the pseudorandom function evaluated at x using the key k), while the server learns nothing.

This fulfills the security properties expected from an OPRF that, as a secure two-party protocol, has the goals of privacy and obliviousness.

It is also important to note that there are many different constructions of OPRFs in the literature and they have been listed based on their underlying PRF and high-level method of oblivious evaluation [32].

One common OPRF construction is the Hashed Diffie-Hellman OPRF (HashDH), we will be exploring this construction in more detail in Chapter 4. This OPRF uses a a very simple "blinded exponentation" protocol depicted in the figure below:

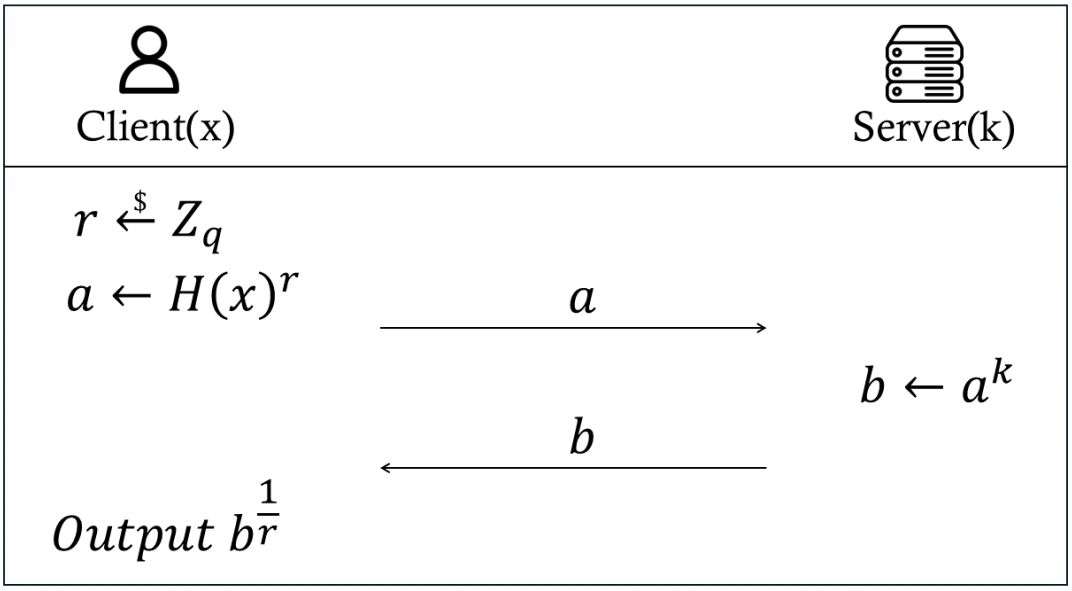

Figure 2.2: Blinded exponentiation for evaluating the HashDH PRF.

In this technique for evaluating the HashDH PRF the client starts by selecting a random blinding factor r chosen uniformly at random. Then the client computes a blinded version of its input. This effectively masks the actual input value, after this the client sends the blinded input to the server.

The key idea behind this construction is that the blinding factor applied by the client ensures that the server cannot determine the actual input value from the blinded input it receives. Additionally, the server computes the PRF output without revealing his secret key to the client.

2.4.2.1 Verifiable Oblivious Pseudorandom Functions (VOPRFs)

VOPRs are an extension of OPRFs that add a verifiability property by enabling the server to provide a cryptographic proof of the correctness of the computation. This allows the client to verify that the server has evaluated the PRF correctly with respect to a prior commitment to the key. So, basically a verifiable OPRF (VOPRF) is an OPRF where the server also proves to the client that $F(k, x)$ was computed with the secret key k corresponding to the server's public key, which the client knows [42].

In a VOPRF protocol, the client and server engage in a cryptographic interaction to compute the output of the PRF on the client's input. The key addition in a VOPRF is that the server provides a proof, often in the form of a zero-knowledge proof or a non-interactive proof, to convince the client that it has correctly computed the output without revealing any information about the client's input.

Zero-Knowledge Proofs: Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that allow one party (the prover) to convince another party (the verifier) that a statement is true without revealing any information beyond the validity of the statement itself. In other words, the prover can demonstrate knowledge of a secret (e.g., a cryptographic key or an internal computation) without actually revealing the secret itself. This is essential for VOPRFs, where the server proves it correctly computed $F(k, x)$ without disclosing the secret key k or any internal details.

Formally introduced by Goldwasser, Micali, and Rackoff [61], zero-knowledge proofs must satisfy three properties:

- **Completeness**: If the statement is true, an honest verifier will be convinced by the proof.
- **Soundness**: If the statement is false, a dishonest prover cannot convince the verifier that it is true.
- **Zero-Knowledge**: If the statement is true, the verifier learns nothing other than the fact that the statement is true.

Many modern VOPRF constructions rely on Non-Interactive Zero-Knowledge Proofs (NIZKs), which allow the server to generate a proof that the client can verify in a single round, eliminating the need for extra interaction by allowing the prover to generate a proof in advance that the verifier can later verify without needing further communication, improving efficiency in practical implementations.

There is also the notion of VOPRFs to include public verifiability, where any third party can verify the correctness of the computation [19].

VOPRFs have applications in various scenarios like the ones we specified in OPRFs before but especially in scenarios where the verifiability of the server's computation is crucial to maintain trust and security [3].

2.4.2.2 OPRFs Constructions

In this subsection we show in Table 2.1 the list of practically relevant OPRFs from the literature based on their underlying PRF and high-level method of oblivious evaluation.

To gain an overview of the many OPRF constructions in the literature, we first observe that we can sort them into mainly four categories, as discussed by Casacuberta et al. in[32].

PRF	OPRFs	Method
Naor-Reingold	[19, 57, 63, 64]	Oblivious Transfer
	[1, 67]	Homomorphic encryption
HashedDH	[7, 8, 41, 43, 66, 67, 69, 70]	Blinded exponentiation
Dodis-Yampolskiy	[26, 29, 72, 86]	Homomorphic encryption
	[109]	Blinded exponentiation
Any	[73, 75, 77, 78, 96, 97, 98]	Oblivious Transfer

Table 2.1: Classification of OPRF protocols based on their underlying PRF and method of oblivious evaluation

2.4.3 Password-Based Authentication Protocols

Password-based authentication is one of the oldest and most widely used methods of verifying the identity of an user in various systems and applications. It involves the use of a password which is a secret string known only to the user to authenticate their access, the server then compares the provided password with the stored password associated with the user's account to see if it matches.

This authentication method as the objective of ensuring that only authorized users who possess the correct password can gain access to protected systems or information. It relies on the assumption that the password is known only to the legitimate user and not to unauthorized individuals.

Even though this authentication method is very popular, password-based authentication faces a lot of security challenges, such as password brute force attacks, password leakage, and password database breaches.

To enhance security and to deal with this challenges password-based authentication systems often employ additional measures like secure password storage and protection and password-based authentication protocols.

2.4.3.1 Password Storage and Protection

This authentication method needs a way for the server to verify if the user password corresponds to who the user is claiming to be, therefore after a user creates a password, a copy of that password is stored by the system/server in a database.

This is a critical aspect of password-based authentication since storing passwords in plaintext is highly insecure, and exposes them to potential attackers in the event of a data breach. Because of this modern systems employ cryptographic techniques to store passwords securely, for example they use hash functions such as bcrypt, scrypt, and Argon2 to encrypt the passwords stored in the server.

However this just by itself is not sufficient enough to ensure security since these passwords are still vulnerable to dictionary attacks or precomputation table attacks if the attacker gets access to the server database.

2.4.3.2 Password-Based Authentication Protocols

Password based authentication protocols have been developed to enhance security and address some vulnerabilities of password based authentication.

Two notable examples are the Secure Remote Password (SRP) protocol, which provides secure mutual authentication between a client and a server without revealing the password to the server, and the Password Authenticated Key Exchange (PAKE) protocol, which enables secure key establishment solely based on password authentication. These protocols incorporate cryptographic techniques, such as zero-knowledge proofs and challenge-response mechanisms, to prevent various attacks and ensure secure authentication.

2.4.3.3 Password-Authenticated Key Exchange (PAKE)

The PAKE protocol, which was first proposed by Bellovin and Merritt [14], is a special form of the cryptographic key exchange protocol. Key exchange (or "key agreement") protocols are designed to assist two parties (call them the client and the server) in reaching an agreement on a shared key, using public key cryptography. Early key exchange protocols such as the famous Diffie-Hellman were unauthenticated, leaving them open to man-in-the-middle attacks.

The distinctive feature of the PAKE protocol is that it provides key exchange capabilities without requiring additional authentication factors, such as tokens or biometrics. Instead, the authentication relies solely on the knowledge of the password by the involved parties (the password or its hash is assumed to be known to the server, allowing for verification). This allows for secure key establishment while leveraging passwords as the shared secret. This way PAKE can provide secure authentication and key establishment in scenarios where users only have a password as their secret authentication credential. Traditional authentication protocols that rely on password-based authentication suffer from various vulnerabilities, such as password guessing attacks, offline dictionary attacks, and interception of passwords during transmission [62].

PAKE protocols address these vulnerabilities by employing cryptographic techniques to protect the privacy and integrity of the password during the authentication process. They ensure that the password is securely transformed into a shared cryptographic key that can be used for subsequent secure communication between the client and the server.

What makes a PAKE truly useful is that it should also provide protection for the client's password. A stronger version of this guarantee can be stated as follows: after a login attempt (valid, or invalid) both the client and server should learn only whether the client's password matched the server's expected value, and no additional information.

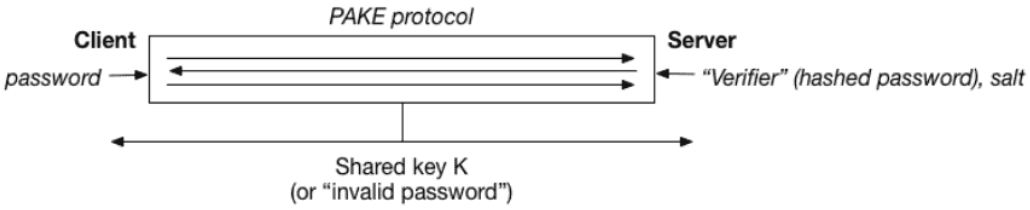

Figure 2.3: Ideal representation of a PAKE protocol. The two parties inputs also include some randomness, which isn't shown. An eavesdropper should not learn the strong shared secret key K, which should itself be random and not simply a function of the password [62].

2.4.4 OPAQUE Protocol

OPAQUE is a cryptographic protocol designed to enhance the security of password-based authentication. It aims to protect user passwords even in the presence of server compromise or offline attacks. The protocol was proposed and formally analyzed by Stanislaw Jarecki, Hugo Krawcyzk and Jiayu Xu in 2018 [69].

The primary goal of OPAQUE is to provide password-authenticated key exchange (PAKE) functionality, allowing two parties to establish a secure shared key over an insecure channel using only a password as a shared secret. The protocol is oblivious in the sense that it ensures that no information about the password is leaked during the authentication process.

OPAQUE provides security against offline dictionary attacks by employing a verifiable oblivious pseudorandom function (OPRF). An OPRF allows the client to generate a proof that it has evaluated the function correctly on its password, without revealing the actual password itself. This ensures that an attacker with access to the server's data cannot determine the user's password.

By combining these techniques, OPAQUE offers a robust and secure solution for password-authenticated key exchange. It provides strong security guarantees, even in the face of server compromise or offline attacks on stored password data.

It is worth noting that OPAQUE has undergone several revisions and refinements since its introduction, and different variants of the protocol have been proposed to address specific security requirements or deployment scenarios. Nonetheless, the core principles and goals of the OPAQUE protocol remain consistent across these variations.

The IRTF define OPAQUE as "A secure asymmetric password-authenticated key exchange (aPAKE) that supports mutual authentication in a client-server setting without reliance on public key infrastructure (PKI) and with security against pre-computation attacks upon server compromise. In addition, the protocol provides forward secrecy and the ability to hide the password from the server, even during password registration" [21].

2.4.4.1 OPAQUE protocol Overview

In this section we will look at a general overview of OPAQUE. First we provide a more formal and technical overview of OPAQUE protocol according to the draft from the IRTF(Internet Research Task Force) [21] and after we give a more intuitively overview of the OPAQUE protocol. OPAQUE consists of two stages:

- **Stage 1 : Registration** - In this first stage the client registers its password with the server and stores its credential file on the server.

- **Stage 2 : Authenticated key exchange** - After we have the second stage also known as the "login" stage where the client recovers its authentication material and uses it to perforam a mutually authenticated key exchange.

However prior to both stages there is a setup phase where the client and server need to agree on a configuration that fully specifies the cryptographic algorithm dependencies necessary to run the protocol, during these phase the server chooses a pair of keys (server private key and server public key) for the AKE, and chooses a seed (oprf seed) of Nh bytes for the OPRF. The server can then use this single pair of keys with multiple clients and can opt to use multiple seeds (so long as they are kept consistent for each client).

During the first stage, also known as the registration stage, the client inputs its credentials, which include its password and user identifier, and the server inputs its parameters, which include its private key and other information. The only client output of this stage is a single value export key that the client may use for application-specific purposes, for example to encrypt additional information for storage on the server. The server does not have access to this export key. On the other side, the server output of this stage is a record corresponding to the client's registration that it stores in a credential file alongside other clients registrations as needed. These messages are the registration request, registration response, and registration record, respectively.

It is important to note that this is the only stage in OPAQUE that requires a server-authenticated channel with confidentiality and integrity: either physical, out-of-band, PKI-based, etc.

Below we can see an image showing the registration flow of stage 1:

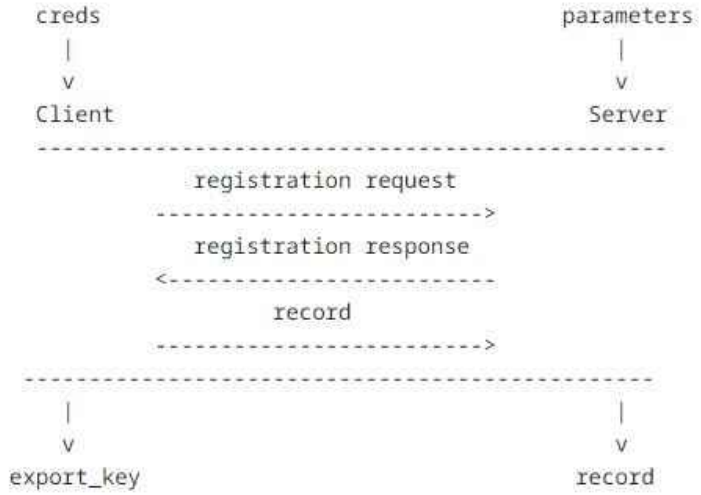

Figure 2.4: Registration flow of stage 1 [21]

Now looking at the second stage, also known as the authenticated key exchange stage. The client starts by obtaining credentials previously registered with the server, recovers private key material using the password, and subsequently uses them as input to the AKE protocol. As in the registration phase, the client inputs its credentials, including its password and user identifier, and the server inputs its parameters and the credential file record corresponding to the client. The client outputs two values, an export key (matching that from registration) and a session key, the latter of which is the primary AKE output. The server outputs a single value session key that matches that of the client. Upon completion, clients and servers can use these values as needed. These messages carry the messages of the concurrent execution of the key recovery process (OPRF) and the authenticated key exchange (AKE).

Below we can see an image showing the authenticated key exchange flow of stage 2:

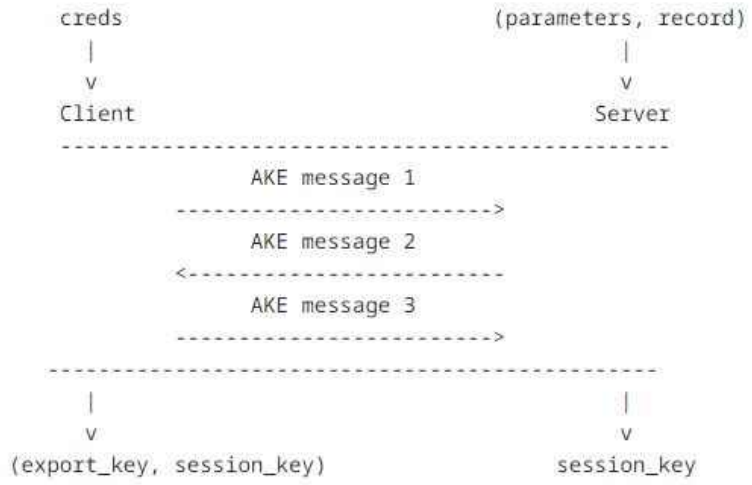

Figure 2.5: Authenticated key exchange flow of stage 2 [21]

More intuitively OPAQUE protocol enables to store user secrets on a server, without giving the server access to those secrets. This is done by storing a secret envelope in the server that is "locked" by two pieces of information: the client's password known only by the client, and a random secret key (like a salt) known only by the server instead of storing a traditional salted password hash. After that, to log in the client initiates a cryptographic exchange that reveals the envelope key to the client, but not to the server.

After this the server then sends the envelope to the user, who now can retrieve the encrypted keys. These keys, once unlocked, will be the inputs to an Authenticated Key Exchange (AKE) protocol, which allows the user and server to establish a secret key which can be used to encrypt their future communication [22].

To conclude, the user first signs up for a service picking a username and password before the registration stage. Then the registration stage begins with the client and the server engaging in an OPRF exhcange, the result is that the client has a random key *rkey*, derived from the OPRF output $F(kU, pwdU)$, where kU is a server-owned OPRF key specific to the client and $pwdU$ is the client's password. Within its OPRF message, the server sends the public key for its OPAQUE identity. The client then generates a new private/public key pair, which will be its persistent OPAQUE identity for the server's service, and encrypts its private key along with the server's public key with the *rkey* (this results in an encrypted envelope that are the client's credentials). The client then sends its credentials along with its public key (unencrypted) to the server, who stores the data the client provided, along with the client's specific OPRF keysecret, in a database indexed by its username.

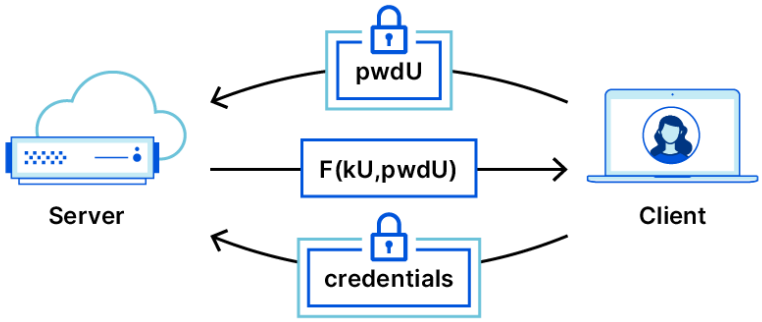

Figure 2.6: Registration phase [22]

Then we have the next stage, the login or authenticated key exchange stage, it also starts with an OPRF flow. However, this time on the server side, instead of generating a new OPRF key, the server instead looks up the one it created during the client's registration. The server does this by looking up the client's username (which the client provides in the first message), and retrieving its record of the client. This record contains the client's public key, the credentials envelope, and the server's OPRF key for the client.

The server also sends over the credentials which the client can decrypt with the output of the OPRF flow. (If decryption fails, the client aborts the protocol, this likely indicates that the client typed its password incorrectly, or that the server isn't who it says it is). If

decryption succeeds, the client now has its own secret key and the server's public key. The client inputs these into an AKE protocol with the server, who, in turn, inputs its private key and the client's public key, which gives them both a fresh shared secret key.

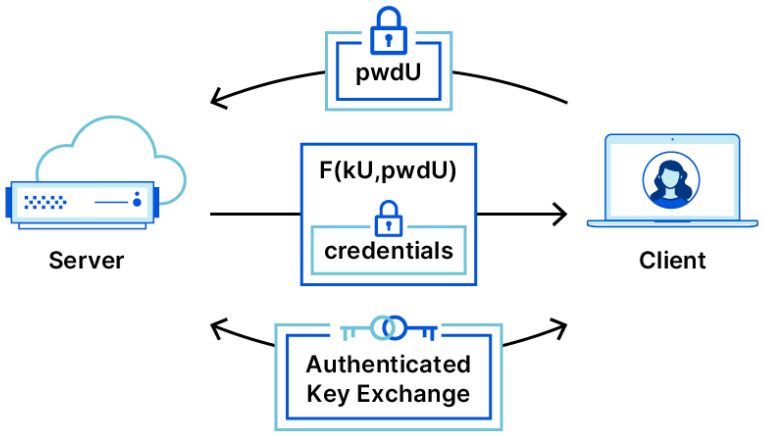

Figure 2.7: Login phase [22]

2.4.4.2 Security Considerations in OPAQUE

The OPAQUE protocol introduces several security considerations to ensure the robustness of password-based authentication. Int this section we aim to explore and enumerate these security aspects, emphasizing the strengths and potential vulnerabilities of the OPAQUE protocol.

These are some of the security considerations that we have to take into account when talking about OPAQUE:

- **Password Security:** One crucial aspect of OPAQUE is the protection of user passwords. The protocol employs verifiable oblivious pseudorandom functions (OPRFs) to safeguard password privacy during authentication. This ensures that the server does not gain knowledge of the actual password, even in the presence of server compromise or offline attacks. By preventing offline dictionary attacks, OPAQUE significantly enhances password security.

- **Server Security:** The security of the server storing password verifiers and performing authentication is of paramount importance. OPAQUE addresses this concern through the use of salted password verifiers. By incorporating salts, which are unique per user, OPAQUE protects against pre-computation attacks and rainbow table attacks. Even if an attacker gains access to the server's data, the salts and hashed verifiers make it computationally infeasible to retrieve the actual passwords.

- **Privacy Preservation:** OPAQUE takes privacy preservation seriously during the authentication process. By employing Diffie-Hellman key exchange, the client and

server can establish a shared secret without disclosing their private keys or the password itself. This ensures that eavesdroppers cannot gain any information about the password or the exchanged keys. The use of public and private keys in the protocol contributes to preserving the privacy of both parties.

- **Resistance to Attacks:** The OPAQUE protocol demonstrates resilience against various types of attacks. It is designed to withstand offline attacks, including precomputation attacks and password cracking attempts. By employing secure hashing algorithms and OPRFs, OPAQUE ensures that even if an adversary gains access to stored data, they cannot derive the user's password. Additionally, the Diffie-Hellman key exchange component provides resistance against man-in-the-middle attacks.

- **Cryptographic Assumptions:** The security of the OPAQUE protocol relies on certain cryptographic assumptions. Specifically, the protocol assumes the security of the Diffie-Hellman assumption and the OPRFs used. Any vulnerabilities or limitations in these assumptions could have implications for the overall security of the protocol. Therefore, ongoing research and scrutiny of these underlying cryptographic assumptions are essential to ensure the long-term security of OPAQUE.

- **Side-Channel Attacks and Implementations:** While the OPAQUE protocol offers strong security guarantees, it is crucial to consider potential side-channel attacks that could compromise its security. Side-channel attacks exploit information leaked through physical implementations or timing measurements. Implementers must adhere to best practices, such as constant-time implementations and countermeasures against power analysis attacks, to mitigate these risks and ensure the protocol's resilience in real-world deployments.

2.5 Summary and Critical Analysis

This chapter has provided an overview of the fundamental principles of modern cryptography, focusing on cyrptographic keys and their generation and into the cryptographic primitives and protocols that are related to OPRFs.

A critical examination of the related work reveals a key issue that lies at the heart of this thesis: the reliance on high-entropy key generation to maintain the security of cryptographic protocols. As emphasized in several sections, strong, unpredictable keys are essential for ensuring the security guarantees provided by OPRFs and related applications. However, the possibility of generating weak keys due to insufficient randomness, whether due to a malicious or compromised server, poses a significant risk to the overall security of these protocols.

The literature suggests that the security of OPRF is highly sensitive to the quality of the key generation process, a critical dimension in the context of key rotation. This will be something that we will investigate in this thesis.

Furthermore, the analysis of certain primitives and assumptions in the literature, such as dual PRFs, will be particular interesting when exploring whether alternative mechanisms can be developed to strengthen trust guarantees for clients, even when the server's key generation process is potentially compromised. The ability of these constructs to incorporate both the key and the input in generating pseudorandom outputs offers a potential path toward greater trust and security in scenarios where the server's key rotation is unreliable.

This critical analysis serves as the basis for the following chapters, which will focus on investigating the specific challenges and potential solutions surrounding key rotation security in OPRF protocols, particularly in the context of a compromised server that can no longer derive truly random keys.

It is also important to define that in this thesis we will be using the following notation.

Notation: We denote the security parameter as λ. For any $\lambda \in \mathbb{N}$, let 1^λ be the unary representation of λ. We write $x \xleftarrow{\$} \mathcal{S}$ to indicate that we choose an element x uniformly at random from set $\mathcal{S}$. Unless explicitly stated otherwise, all logarithms below are base 2.

3

Analyzing PRF Security

In our pursuit of a deeper understanding of Oblivious Pseudo-Random Functions (OPRFs) and the role played by the keys in these protocols, we start by focusing on the heart of OPRF functionality: the pseudorandom function (PRF).

Therefore, in this chapter we present our formal definition of a PRF and we define our security model, an instrument that will be essential for analyzing the security of the PRF.

3.1 PRF Security Model

As we discussed previously, in modern cryptography the scrutiny and analysis of cryptographic primitives, such as pseudorandom functions (PRFs), is grounded in well-defined security models. These models serve as essential instruments for evaluating the resilience of cryptographic constructions, through a formalized set of rules and criteria these security models not only delineate the security goals and operational boundaries of cryptographic primitives but also provide a systematic framework for identifying and addressing potential vulnerabilities.

In this chapter, we will introduce our security model. We will start with formally defining the pseudorandom function (PRF), accompanied by the formal definitions of distributions for key and input selection, the security game and the adversary model. We will follow the principles of modern cryptography, providing formal definitions, precise assumptions and proofs of security.

3.1.1 Defining the PRF

In Chapter 2 we provided a brief introduction to the concept of a pseudorandom function (PRF) but in this chapter we will go into a more careful analysis and will define the Standard PRF security model. A PRF is a keyed function that behaves essentially like a random function, to be more precise, a pseudorandom function (PRF) is a two-input function that takes as input a secret key k and an input x producing an output that appears indistinguishable from a truly random function.

This is the primary security goal of a PRF: to be a "random-looking" function. But like Katz and Lindell [74] said, it makes little sense to say that any fixed function $f : \{0,1\}^* \to \{0,1\}^*$ is pseudorandom, we must instead refer to the pseudorandomness of a distribution on functions, such a distribution is induced naturally by considering keyed functions.

Definition 8 (Keyed function)**.** *A keyed function F is a two-input keyed function $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$ where the first input is called the key and denoted k and where the second input is denoted by x.*

A keyed function F induces a natural distribution on functions given by choosing a uniform key $k \in \{0,1\}^\lambda$ and then considering the resulting single-input function F_k. We call F pseudorandom if the function F_k (for a uniform key k) is indistinguishable from a function f_r chosen uniformly at random from the set of all functions having the same domain and range. In the formal definition bellow we will be using and defining the following libraries: $\mathcal{L}^F_{\text{prf-real}}$ representing the PRF and $\mathcal{L}^F_{\text{prf-rand}}$ representing a function chosen uniformly at random from the set of all functions having the same domain and range.

We say that F is efficient if there is a polynomial-time algorithm that computes $F(k, x)$ given k and x. In this security model we will assume this since we are only interested in efficient keyed functions.

We will be using the security parameter λ, it dictates the key length, input length, and output length regarding the PRF.

3.1.2 Formal Definition

This is the standard PRF definition with the aim of expressing a standard PRF security model. In this initial section of the security model, we will formally define the key sammpling and input selection for the PRF and present a formal definition of the PRF.

3.1.2.1 Key generation

Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of distributions parameterized by a security parameter λ

The key k is then sampled according to the distribution $\mathcal{D}_\lambda$:

$$k \leftarrow \mathcal{D}_\lambda$$

In most cryptographic constructions, including the security definitions for pseudorandom functions, it is assumed that the distribution $\mathcal{D}_\lambda$ is the uniform distribution over the key space. Thus, k is chosen uniformly at random from the key space $\{0,1\}^\lambda$. This assumption ensures that the key k is chosen uniformly at random, which maximizes the entropy of the key, this is critical since a high-entropy key makes it extremely difficult to predict, which ensures strong security properties in cryptographic protocols. The more entropy a key has, the more resistant it is to attacks such as brute-force, where an attacker

tries all possible keys, or any other method that tries to predict the key based on patterns. This uniform key distribution underpins the indistinguishability property that defines the security of the PRF.

3.1.2.2 Input selection

In the standard PRF security model, the input x is typically chosen by the adversary as part of their queries to the PRF oracle. This is a key aspect of PRF security: the adversary can choose specific inputs $x \in \{0,1\}^\lambda$ and query the oracle.

For each new query that the adversary makes, he chooses an $x \in \{0,1\}^\lambda$.

3.1.2.3 Libraries

We will use two libraries, one to represent the pseudorandom function and one to represent a function chosen uniformly at random from the set of all functions having the same domain and range.

- $\mathcal{L}^F_{\text{prf-real}}$ represents a PRF.
- $\mathcal{L}^F_{\text{prf-rand}}$ represents a function chosen uniformly at random from the set of all functions having the same domain and range.

Choosing a function uniformly at random from the set of all functions having the same domain and range might not sound very intuitive. We can look at this random function as a uniformly chosen array/lookup table and it can be accessed through the *lookup* subroutine of the following library:

$\mathcal{L}^F_{\textbf{prf-rand}}$

T := empty associative array

lookup($x \in \{0,1\}^\lambda$):

---

if $T[x]$ is undefined:
$T[x] \leftarrow \{0,1\}^\lambda$
return $T[x]$

When the lookup function is called by the adversary (*lookup*(x)), the PRF function F takes the key k and the input x and produces an output $F(k,x)$. This function is indistinguishable from the function PRF-rand, meaning that when the adversary performs the *lookup* they do not know if they are interacting with the PRF function or with a random function. Therefore, we call F pseudorandom if the function F_k (for a uniform key k) is indistinguishable from a function chosen uniformly at random from the set of all functions having the same domain and range, in other words, if no efficient adversary can distinguish

whether it is interacting with F_k (for uniform k) or f (where f is chosen uniformly from the set of all functions mapping λ-bit inputs to λ-bit outputs).

More formally, the following library should be indistinguishable from the one above:

$\mathcal{L}^F_{\textbf{prf-real}}$

$k \leftarrow \mathcal{D}_\lambda$

lookup($x \in \{0,1\}^\lambda$):

return $F(k, x)$

Definition 9 (PRF Security)**.** *Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a deterministic function.*

We say that F is $\mathcal{D}$-secure if $\mathcal{L}^F_{prf\text{-}real} \approx \mathcal{L}^F_{prf\text{-}rand}$, where $\approx$ denotes computational indistinguishability, meaning that no efficient (polynomial-time) algorithm can distinguish between the two distributions with a non-negligible probability.

However it's important to make a note regarding these two libraries that we are using: We want to build our security definitions with libraries that run in probabilistic polynomial time (the rationale behind this is explained in Subsection 2.1.3), because of this no matter how big the table T is meant to be, a polynomial-time calling program will only access a polynomial amount of it. This means that T initially starts uninitialized, and its values are only assigned as the calling program requests them. This changes when each T [x] is sampled (if at all), but does not change how it is sampled (i.e., uniformly and independently).

3.1.3 Adversary Model

In this subsection, we define the adversary model, detailing the capabilities and limitations of the adversary in attacking the PRF, and clarifying the security goals.

We create a notion of a probabilistic-polynomial time adversary $\mathcal{A}$ that can query an oracle O which is either equal to the behaviour of $\mathcal{L}_{\text{prf-real}}$ or $\mathcal{L}_{\text{prf-rand}}$.

The adversary $\mathcal{A}$ may query its oracle at any point $x \in \{0,1\}^\lambda$ in response to which the oracle returns $O(x)$. The oracle computes a deterministic function and so it returns the same result if queried twice on the same input, for this reason we may assume without loss of generality tha $\mathcal{A}$ never queries the oracle twice on the same input.

$\mathcal{A}$ can interact freely with its oracle, choosing its queries adaptively based on all previous outputs. However since $\mathcal{A}$ runs in polynomial time it can ask only polynomially many queries.

Formally we say that PRF is secure if, for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function *negl* such that:

$$\text{Adv}_{\text{PRF}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathcal{L}^F_{\text{prf-real}}}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathcal{L}^F_{\text{prf-rand}}}(1^\lambda) = 1]\right| \leq \text{negl}(\lambda)$$

where the first probability is taken over the randomness of $\mathcal{A}$, the randomness used in the sampling of k. And the second probability is taken over the randomness of $\mathcal{A}$, and by the random choice of $\mathcal{L}_{\text{prf-rand}}$ choosing a random function by uniformly sampling its truth table as needed(there are $2^{\lambda \times 2^{\lambda}}$ possible functions for T). In simpler terms, if an adversary cannot distinguish with non-negligible advantage whether they are interacting with the PRF or a truly random function, then the PRF is considered secure.

It is also very important to understand that $\mathcal{A}$ does not know k. If k is revealed any claims about the pseudorandomness of the PRF no longer hold since all the adversary has to do is query the oracle at any point x to obtain the answer y, and compare this to the result $y' := F_k(x)$ that it computes itself using the known value k. An oracle for F_k will return $y = y'$, while an oracle for a random function will have $y = y'$ with probability only $2^{-\lambda}$.

3.1.4 Security Game

We can also define this PRF by the notion of a security game.

Security games are a fundamental tool in cryptography for formally defining and evaluating the security of cryptographic primitives and protocols. These games provide a structured framework to analyze how well a system or primitive meets its security goals under various conditions and potential attacks. We present a security game definition in this subsection to help formalize the security properties and to help model potential adversarial behavior.

PRF security game:
Let $F : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$. The security game is played between an adversary $\mathcal{A}$ and a challenger. The game is parameterized by a bit $b \in \{0,1\}$. This is like a coin flip that the challenger does that chooses between the real game($b = 0$) or the random game($b = 1$).

1. If $b = 0$, the challenger samples a key $k \leftarrow \mathcal{D}_{\lambda}$ and sets $f = F(k, .)$, meaning that he will be answering to the adversary with $\mathcal{L}_{\text{prf-real}}$. If $b = 1$, the challenger samples a uniformly random function f from the set of all functions with same domain and range, formally $f \xleftarrow{\$} \mathcal{F}_{\{0,1\}^{\lambda} \to \{0,1\}^{\lambda}}$, meaning that he will be answering with $\mathcal{L}_{\text{prf-rand}}$.

2. The adversary chooses an $x \in \{0,1\}^{\lambda}$ for each new query and sends it to the challenger.

3. The challenger replies with $f(x)$.

4. The adversary can continue to submit polynomially many queries to the challenger (repeating steps 2 and 3) and at the end of the game the adversary outputs bit $b' \in \{0,1\}$ as a guess (whether it believes it is interacting with F ($\mathcal{L}_{\text{prf-real}}$) or a random function($\mathcal{L}_{\text{prf-rand}}$).)

The adversary $\mathcal{A}$ wins the game if $b' = b$.

The PRF F is considered secure if for all efficient adversaries $\mathcal{A}$, the probability of winning the game (correctly distinguishing if it is interacting with $\mathcal{L}_{\text{prf-real}}$ or $\mathcal{L}_{\text{prf-rand}}$) is negligible. Formally, we require that:

$$\Pr[b' = b] \leq \frac{1}{2} + \text{negl}(\lambda)$$

where $\text{negl}(\lambda)$ is a negligible function in the security parameter λ.

This expresses that the probability of the adversary guessing the correct bit(denoted $\Pr[b' = b]$) should not significantly exceed $\frac{1}{2}$ plus some negligible factor determined by λ. This ensures that the adversary cannot do significantly better than random guessing, indicating the security of the PRF.

4

OPRF Construction

Now that we have the security model defined we can focus on an actual concrete cryptographic construction.

In this chapter we focus on the 2HashDH OPRF construction, and on its underlying PRF, $H(x)^k$, providing formal proof that it is indeed a standard PRF and that it fits in our PRF security model, subsequently we analyze the security impact when the key is sampled from a non-uniform distribution.

4.1 OPRF Construction

We want to study a concrete PRF construction so that we can analyze more concretely the impact of the key in the security of the PRF and consequently in the security of the OPRF. In this case we will look into the 2HashDH OPRF construct(a slight variation of the HashDH OPRF).

There are various OPRF constructions in literature as we illustrate in Subsection 2.4.2.2. In this thesis we choose the 2HashDH OPRF construction for our analysis because, as it has been shown by Jarecki et al. [7, 66, 67, 69], it can be proven secure in the Universal Composability (UC) framework [30]. 2HashDH is a relevant, highly practical construction that is used extensively in the literature [7, 8, 41, 43, 66, 69], it has also been demonstrated that this construction is versatile in terms of properties since the protocol allows for verifiable computation through efficient non-interactive zero knowledge proofs (NIZK), or a threshold version for sharing the OPRF key among multiple servers.

Lastly, the 2HashDH construction is secure under the One-More Gap Diffie-Hellman (OM-gapDH) assumption and with both hash functions modeled as random oracles. This means that the security of the protocol can be proven based on the difficulty of solving the OM-gapDH problem, with the additional assumption that the hash functions used in the construction behave like ideal random oracles. We will go further into the one-more gap Diffie-Hellman (OM-gapDH) assumption in the next subsection since the security of this specific construct is formally proven based on this very important assumption.

4.1.1 Hashed Diffie-Hellman(HashDH) Construction

The 2HashDH OPRF construction is a slight variation of the HashDH construction, so to understand better the 2HashDH construction we will first look at the HashDH construction.

The underlying PRF in the HashDH OPRF is the function $f_k^H(x) := H(x)^k$ with hash function H. It is a PRF under the the idealized assumption that H produces uniformly random elements from a group $\langle g \rangle$ [89], this means that H is a hash function that maps inputs to uniformly random elements from $\langle g \rangle$ which means that for any input x, $H(x)$ can be represented as g^a for some random $a \in \mathbb{Z}_q$ due to cyclic group properties. Implicitly when setting $g^a \leftarrow H(x), f_k^H(x) = (g^a)^k = g^{ak}$ becomes a Diffie-Hellman value and this is why this PRF is refered as HashDH. f^H can be obliviously evaluated with a blinded exponentiation protocol, as shown below, which has been shown to be secure under the one-more gap Diffie-Hellman(OM-gapDH) assumption in the random oracle model(ROM)[67]. It is relevant to note that multiplicative blinding instead of blinded exponentiation has been considered for HashDH since the first one requires only fixed-base exponentiation and hence decreases the client's computational cost, however this has been very recently investigated by Jarecki et al. [70] and the conclusion is that the resulting protocol cannot satisfy standard OPRF security notions and is recommended to be used only when the correct value of the public key g^k is authenticated, and when the OPRF inputs are of high entropy. For this reason, in this thesis we will consider HashDH with blinded exponentiation.

Blinded exponentiation protocol in the context of HashDH:

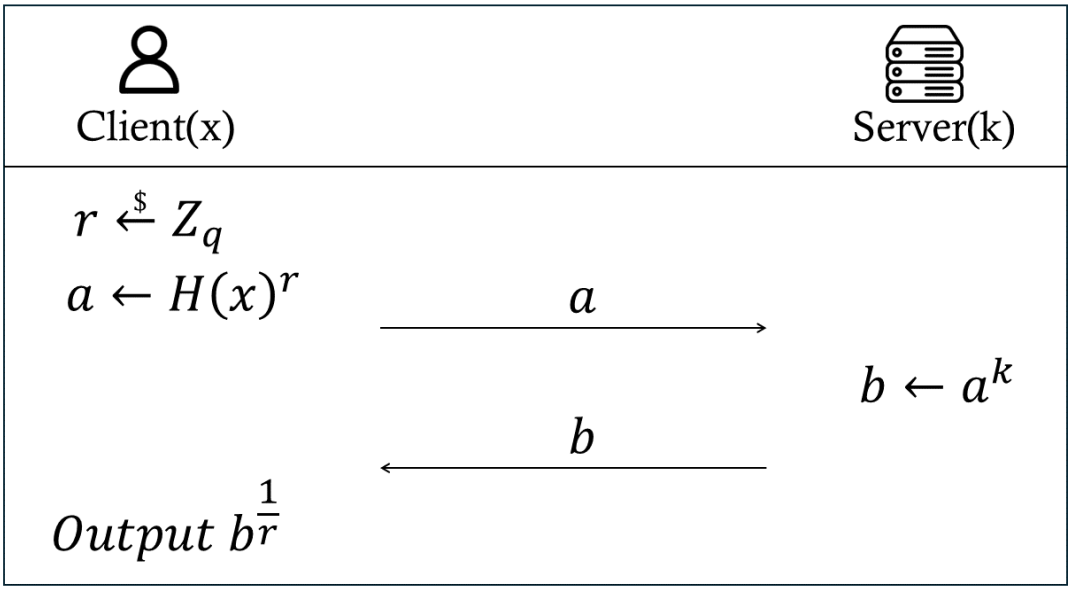

Figure 4.1: Diagram of the HashDH OPRF construction

- **Client**: Wants to compute $H(x)^k$ without revealing x to the server.

- **Server**: Holds a secret key k and helps in the computation without learning x or revealing k to the client.

1. **Client blinds the input:**
   - The client has an input x.
   - The client chooses a random blinding factor $r \leftarrow \mathbb{Z}_q$, where q is a large prime.
   - The client computes the blinded input $a = H(x)^r$, the security assumption of the discrete logarithm problem in the group $\mathbb{Z}_q$ ensures that the server cannot compute x from $H(x)^r$.
2. **Client sends the blinded input:**
   - The client sends $a = H(x)^r$ to the server.
3. **Server's computation:**
   - The server receives a and computes $b = a^k$.
   - Mathematically: $b = (H(x)^r)^k = H(x)^{rk}$.
4. **Server sends the result:**
   - The server sends b back to the client.
5. **Client unblinds the result:**
   - The client receives $b = H(x)^{rk}$ and unblinds it by computing $b^{1/r}$.
   - Mathematically: $b^{1/r} = (H(x)^{rk})^{1/r} = H(x)^k$.

The idea behind this protocol, Blinded DH exponentiation, has been a known idea in cryptography for some time, it has been used for example for building blind signatures [36].

4.1.2 2HashDH Construction

The 2HashDH is a slight variation of the HashDH OPRF, its construction is also rooted in the idea of blinded DH exponentiation and it forms the core of many protocols aiming to achieve secure and oblivious interactions between parties. In this variation the client adds an outer hash H_2 and outputs $f^{2H} := H_2(x, H_1(x)^k)$.

Like we mentioned earlier, the 2HashDH construction allows for verifiable computation through efficient non-interactive zero knowledge proofs (NIZK) making it effectively a VOPRF protocol where the client can verify that the server has evaluated the PRF with the key k corresponding to the server's public key.

In 2HashDH, the client first hashes and blinds his input and requests the server's secret key application on this blinded value, like in HashDH. The client then verifies the server's

response and obtains the VOPRF output by applying a second hash function. This double hashing action (which is essential in the security proof) is why these type of construction has the "2Hash" prefix in its name.

To be exact, we will take into account the VOPRF 2HashDH construction presented by Jarecki et al. in [67] that has the following structure:

- **Client**: Wants to compute $H_1(x)^k$ (which is part of the VOPRF output $H_2(\pi, x, H_1(x)^k)$) without revealing x to the server. The client also wants to verify that the server is honestly using its secret key k without knowing k itself.
- **Server**: Holds a secret key k and helps in the computation of $H_1(x)^k$ without learning x or revealing k to the client. Additionally, the server provides a Non-Interactive Zero-Knowledge (NIZK) proof to convince the client that the computation was performed correctly using the key k.

1. **Client blinds the input:**
   - The client has an input x.
   - The construction relies on a group of prime order q (where q is a large prime) with a generator denoted g. The client chooses a random blinding factor from this group, $r \leftarrow \mathbb{Z}_q$.
   - The client blinds the input $a = H_1(x)^r$, where H_1 is a hash function that maps inputs to uniformly random elements from $\langle g \rangle$. The security assumption of the discrete logarithm problem in the group $\mathbb{Z}_q$ ensures that the server cannot compute x from $H_1(x)^r$.
2. **Client sends the blinded input:**
   - The client sends $a = H_1(x)^r$ to the server.
3. **Server's computation:**
   - The secret key k is chosen at random from $\mathbb{Z}_q$, and the public key is y is set as $y = g^k$.
   - The server receives a and after checking that $a \in \langle g \rangle$ computes $b = a^k$.
4. **Server sends the result:**
   - The server sends the pair (y, b) to the client.
5. **Client unblinds the result:**
   - The client receives $b = H_1(x)^{rk}$ and unblinds it by computing $b^{1/r}$.
   - The client then computes the VOPRF output as $H_2(\pi, x, H_1(x)^k)$, where $\pi = (g, q, y)$ and H_2 is a hash function onto $\{0,1\}^\lambda$ where λ is a security parameter.

6. **Verification (NIZK Proof):**

   - The client verifies that the tuple $\langle g, y, a, b \rangle$ is a valid Decisional Diffie-Hellman (DDH) tuple.
   - If a $\mathrm{DH}_{g,y}(\cdot,\cdot)$ oracle is available, the client can directly test if $\langle g, y, H_1(x), b^{1/r} \rangle$ is a valid DDH tuple.
   - If such an oracle is not available, the server provides a Non-Interactive Zero-Knowledge (NIZK) proof for equality of discrete logarithms to show that $\log_g y = \log_a b$.
   - Specifically, the server:
     - Selects a random value $t \leftarrow \mathbb{Z}_q$.
     - Computes $w = H_3(g, y, a, b, g^t, a^t)$, where H_3 is a cryptographic hash function.
     - Computes $s = t + w \cdot k \mod q$.
     - Sends the proof $\zeta = (w, s)$ to the client, denoted as $\mathrm{NIZK}^{\mathrm{EQ}}_{H_3}[g, y, a, b]$.
   - The client verifies ζ by testing if $w = H_3(g, y, a, b, g^s y^{-w}, a^s b^{-w})$.
   - If the check passes, the client is assured that the tuple $\langle g, y, a, b \rangle$ satisfies $\log_g y = \log_a b$, meaning the server has correctly used the key k in its computations.

4.1.3 The One-More Gap Diffie-Hellman (OM-gapDH) assumption

We presented in the related work, in Subsection 2.4.1, the traditional Diffie-Hellman assumptions and explained how important they are in cryptography by reducing the security of protocols to the hardness of a DH problem and providing a formal basis for proving the security of cryptographic protocols.

The One-More Gap Diffie-Hellman (OM-gapDH) assumption is a stronger assumption that extends the traditional Diffie-Hellman assumptions. Below we will present informally and formally the One-More Diffie-Hellman (OMDH) assumption, and the One-More Gap Diffie-Hellman (OM-gapDH) that builds on the OMDH and DDH assumptions [46].

- **One-More Diffie-Hellman (OMDH)**: The One-More Diffie-Hellman assumption [9] is a stronger variant of the Diffie-Hellman problem which asserts that the DH problem is hard even with the ability to compute many Diffie-Hellman values(by having acess to a Diffie-Hellman oracle O_{DH}), since an adversary cannot compute one more such value than they have queried.

  Formally, let $(\mathbb{G}, q, g) \leftarrow \mathrm{KeyGen}(\lambda)$ be the key generation algorithm that outputs a multiplicative group of order q and assume $x \leftarrow \mathbb{Z}_q$. We say that the One-More-DH problem is $(\mathrm{negl}(\lambda), t)$-hard if for every algorithm $\mathcal{A}$ that runs in time t we have:

$$\Pr\left[\{(g_i, (g_i)^x)\}_{i=1,\cdots,v+1} \leftarrow \mathcal{A}^{O_{\mathrm{DH}}}(g_1, \cdots, g_{ch})\right] \leq \mathrm{negl}(\lambda)$$

where $ch > v$ and $\mathcal{A}$ made at most v queries to the O_{DH} oracle.

- **One-More Gap Diffie-Hellman (OM-gapDH)**: The One-More Gap Diffie-Hellman assumption [71] is even stronger since it assumes the adversary has access to both a Diffie-Hellman oracle and a decisional Diffie-Hellman oracle.

  Formally, we say that the One-More Gap Diffie-Hellman problem is hard if One-More Diffie-Hellman problem is hard even when the adversary has access to a decisional Diffie-Hellman oracle O_{DDH}:

  $$\Pr\left[\{(g_i, (g_i)^x)\}_{i=1,\ldots,v+1} \leftarrow \mathcal{A}^{O_{\text{DH}}, O_{\text{DDH}}}(g_1, \ldots, g_{\text{ch}})\right] \leq \text{negl}(\lambda)$$

  where ch $> v$ and $\mathcal{A}$ made at most v queries to the O_{DH} oracle.

  Basically, the OM-gapDH assumption is a stronger assumption as it encompasses the OMDH assumption and adds the complexity of the DDH problem. If a construction is secure under the OM-gapDH assumption, it implies that the construction is resilient even when faced with the combined difficulties of both assumptions(DDH and OMDH).

The formal security proof of the 2HashDH OPRF construction relies on demonstrating that the adversary's ability to compute one additional Diffie-Hellman value remains negligible. The OM-gapDH assumption provides the necessary computational hardness guarantee to support this proof, ensuring that the OPRF is secure in practical scenarios and playing a crucial role in the security proof of the 2HashDH OPRF construction since it establishes the inherent computational difficulty required to compromise the security of the construction. Because of this, the security of the OPRF can be proven based on the difficulty of solving the OM-gapDH problem, with the additional assumption that the hash functions used in the construction behave like ideal random oracles.

4.2 Studying the underlying PRF

At the core of the HashDH and 2HashDH OPRF construction lies the PRF $H(x)^k$. Understanding the security definition of $H(x)^k$ is essential for analyzing the security of the OPRF construction.

The first question we address is the formal justification for considering $H(x)^k$ as a standard PRF. To do so, we focus on analyzing $H(x)^k$ under the standard PRF definition, assessing its behavior under typical cryptographic assumptions and principles and demonstrating that $H(x)^k$ is a standard PRF.

4.2.1 $H(x)^k$ as a standard PRF

Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. Let H be a hash function that produces uniformly random elements from this group, where $H(x) = g^{h_x}$ where $h_x \in \mathbb{Z}_p$,

and let $k \in \mathbb{Z}_p$ be a secret key chosen uniformly at random. We define the keyed function $f_k(x) = H(x)^k$ with structure $f_k(x) = g^{h_x \cdot k}$.

Theorem 4.2.1 ($H(x)^k$ as a secure PRF)**.** Formally, we say that $f_k(x) = H(x)^k$ is a secure pseudorandom function (PRF) if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage in distinguishing between the function $f_k(x)$ and a truly random function f_R is negligible:

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{A}}(f_k) = \left|\Pr[\mathcal{A}^{f_k(\cdot)} = 1] - \Pr[\mathcal{A}^{f_R(\cdot)} = 1]\right| \leq \mathrm{neg}(\lambda)$$

where:

- $f_k(x) = H(x)^k$ is the pseudorandom function, with $k \in \mathbb{Z}_p$ chosen uniformly at random,
- $f_R(x)$ is a truly random function that outputs uniformly random and independent values in $\mathbb{G}$ for each distinct input x,
- $\mathcal{A}$ is a PPT adversary that queries an oracle and attempts to distinguish whether the oracle is computing $f_k(x)$ or $f_R(x)$.

Basically, we say that $f_k(x)$ is a secure PRF if for any such adversary $\mathcal{A}$, the advantage $\mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{A}}(f_k)$ is negligible in the security parameter λ.

To formally prove this theorem, we will model the security using a game-based approach.

4.2.1.1 Security Game: PRF Security for $H(x)^k$

The security of $H(x)^k$ as a PRF is modeled by a security game between a challenger and an adversary $\mathcal{A}$. The game proceeds as follows:

- **Setup:** The challenger chooses a secret key $k \in \mathbb{Z}_p$ uniformly at random and generates the function $f_k(x) = H(x)^k$.
- **Oracle Access:** The adversary $\mathcal{A}$ is given access to an oracle $\mathcal{O}$ which computes either the PRF $f_k(x)$ or a truly random function $f_R(x)$. The challenger flips a hidden bit $b \in \{0, 1\}$:
  - If $b = 0$, the oracle computes $f_k(x) = H(x)^k$.
  - If $b = 1$, the oracle computes a truly random function $f_R(x)$.
- **Adversary Queries:** The adversary $\mathcal{A}$ can make polynomially many queries to the oracle, each with a different input x. The oracle responds with either $f_k(x)$ or $f_R(x)$, depending on the value of b.

- **Distinguishing Phase:** After querying the oracle, the adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, indicating whether they believe the oracle was computing the PRF ($b' = 0$) or the random function ($b' = 1$).

The adversary wins the game if it correctly guesses the nature of the oracle with a probability significantly better than random guessing (greater than $\frac{1}{2}$). This means that for the adversary $\mathcal{A}$ to win the game, they need to reliably distinguish between the outputs of the PRF $f_k(x) = H(x)^k$ and the outputs of a truly random function $f_R(x)$.

4.2.1.2 Reduction to the DDH Problem

Theorem 4.2.2. Let $f_k(x) = H(x)^k$ be a PRF, where H is modeled as a hash function that maps its input to uniformly random elements from cyclic group $\mathbb{G}$ generated by g. If there exists a polynomial-time adversary $\mathcal{A}$ that distinguishes between $f_k(x)$ and a truly random function with non-negligible advantage, then we can construct a polynomial-time algorithm $\mathcal{B}$ that solves the Decisional Diffie-Hellman (DDH) problem with non-negligible probability.

Proof. To formally prove the theorem above we will show that an adversary that wins the PRF game too often can be turned into an algorithm that breaks an important hardness assumption. To be more specific, we will show that if there exists an adversary $\mathcal{A}$ that can win the PRF game with non-negligible advantage, then we can construct an algorithm $\mathcal{B}$ that solves the DDH problem with non-negligible probability.

We have formally defined the Decisional Diffie-Hellman assumption above. Intuitively, the DDH problem is as follows: Given (g, g^a, g^b, g^c), where $a, b \in \mathbb{Z}_p$ are chosen uniformly at random, determine whether $g^c = g^{ab}$ (the "Diffie-Hellman" tuple) or whether g^c is a random element from $\mathbb{G}$. The DDH assumption states that no efficient algorithm can distinguish between g^{ab} and a random group element g^c.

Reduction:

Suppose $\mathcal{A}$ is an adversary that can distinguish between $f_k(x) = H(x)^k$ and a random function with non-negligible advantage ϵ. We will construct an algorithm $\mathcal{B}$ that solves the DDH problem using $\mathcal{A}$ as a subroutine.

Given a DDH challenge (g, g^a, g^b, g^c), the goal of $\mathcal{B}$ is to determine whether $g^c = g^{ab}$ or whether g^c is a random group element. Algorithm $\mathcal{B}$ will simulate the PRF game for $\mathcal{A}$ as follows:

- **Input Transformation:** $\mathcal{B}$ receives the DDH challenge (g, g^a, g^b, g^c), where g^a corresponds to the hash function output $H(x)$, and g^b corresponds to the secret key k.

  Thus, $\mathcal{B}$ sets $H(x_1) = g^a$ for a fixed input x_1, and the potential output of the PRF, $H(x_1)^k = (g^a)^b = g^{ab}$, corresponds to g^c. g^c is either g^{ab} (a valid PRF output) or a random group element (simulating a random function).

- **Oracle Simulation:** For each query x_i made by $\mathcal{A}$, $\mathcal{B}$ responds as follows:
  - If $x_i = x_1$, $\mathcal{B}$ returns g^c, which simulates the PRF if $g^c = g^{ab}$, or simulates a random function if g^c is a random group element.
  - If $x_i \neq x_1$, $\mathcal{B}$ generates a random group element g^r, simulating the response of a truly random function.
- **Adversary Output:** After receiving responses to its queries, $\mathcal{A}$ outputs a guess b' for whether the oracle was computing the PRF or a random function.
- **Reduction to DDH:** If $\mathcal{A}$ correctly distinguishes the PRF from a random function with advantage ϵ, then $\mathcal{B}$ can use this information to determine whether $g^c = g^{ab}$ or g^c is random. Specifically:
  - If $\mathcal{A}$ guesses that the oracle is computing the PRF, then $\mathcal{B}$ outputs that $g^c = g^{ab}$ (a valid Diffie-Hellman tuple).
  - If $\mathcal{A}$ guesses that the oracle is computing a random function, then $\mathcal{B}$ outputs that g^c is a random element.

Thus, if $\mathcal{A}$ has a non-negligible advantage ϵ in distinguishing the PRF from a random function, then $\mathcal{B}$ can solve the DDH problem with non-negligible advantage.

Since solving the DDH problem is assumed to be computationally infeasible, it must also be infeasible for any polynomial-time adversary to break the PRF security of $f_k(x) = H(x)^k$. Therefore, $f_k(x) = H(x)^k$ is a secure PRF under the DDH assumption.

□

4.3 Investigating the Impact of a Non-Uniform Key

In the previous section, we demonstrated that $H(x)^k$ is a secure PRF under the assumption that the key k is chosen uniformly at random. This assumption is critical for ensuring that the outputs of the PRF are indistinguishable from random group elements, as it prevents the adversary from exploiting any predictable structure in the key. This assumption is fundamental for ensuring the security properties of the PRF and consequently the security of the VOPRF.

As we have seen before, the 2HashDH VOPRF construction ensures that the client can verify the server's honest behavior without learning the server's secret key k. To achieve this, the server provides a Non-Interactive Zero-Knowledge proof (NIZK) that proves the correctness of its computations without revealing k, by allowing the client to confirm that the server used the secret key k corresponding to the public key $y = g^k$. However this only allows the client to ensure that the server is not cheating by using a different key during the computation but it does not verify the quality of the key k.

This leaves the VOPRF protocol and its implementation to the following pitfall: if the server chooses (intentionally or not) a non-uniform key, then from the perspective of the client all checks pass, and he might now, for example, encrypt his data with a weak key.

With the objective to address this problem, in this section we will analyze the impact of a non uniform key and understand what that means to the advantage of an adversary if the k is non uniform.

4.3.1 Adversary's Advantage with Non-Uniform Key

We first define a non-uniform key distribution as such:

Definition 10 (Non-Uniform Key Distribution)**.** *Let p be a large prime, and let $k \in \mathbb{Z}_p$ be a secret key. We say that k is chosen from a non-uniform distribution if its probability distribution deviates from the uniform distribution over $\mathbb{Z}_p$. Specifically:*

- *When the key k is sampled from $\mathbb{Z}_p$ according to a probability distribution D, where some values of k are more likely to be selected than others. Formally, $D(k_i) = \Pr[k = k_i]$, where $D(k_i)$ differs for different values k_i.*
- *When the key k is uniformly sampled from a smaller subset $S \subseteq \mathbb{Z}_p$, such that $|S| \ll p$. The distribution is uniform over the subset S, but since S is much smaller than the original key space $\mathbb{Z}_p$, the min-entropy of the key is significantly reduced.*

Let k be a secret key sampled from a possibly non-uniform distribution over $\mathbb{Z}_p$. The adversary's goal is to distinguish the output of the PRF $f_k(x) = H_1(x)^k$ from that of a truly random function $f_R(x)$.

In a formal cryptographic proof involving non-uniform key distributions, it is essential to quantify the entropy of the key to help evaluate the adversary's advantage. To formalize this, we use min-entropy, as introduced in Subsection 2.2.4, which measures the worst-case predictability of the key. Specifically, the min-entropy $H_\infty(k)$ of the key k is defined as:

$$H_\infty(k) = -\log_2 \left(\max_{k_i \in \mathbb{Z}_p} \Pr[k = k_i] \right).$$

This measures the predictability of the most likely value of k, the higher the min-entropy, the lower the probability of the most likely key value, meaning the key is more unpredictable. If the key is uniformly distributed, as it is assumed in standard security definitions of a PRF, the min-entropy of k will be $H_\infty(k) = \log_2(p)$, representing maximum unpredictability. In typical cryptographic contexts, where the key space size p is related to the security parameter λ (with $p = 2^\lambda$ and λ representing the bit-length of the key), the min-entropy becomes $H_\infty(k) = \lambda$. This reflects the maximum unpredictability of the key, corresponding to λ bits of security.

However, if k is sampled form a non-uniform distribution, then the min-entropy of k will be lower, meaning that certain values of k are more likely to be chosen than

others, and hence the key is more predictable. This creates potential vulnerabilities in the security of the PRF and the OPRF because adversaries can exploit this reduction in entropy to improve their chances of correctly guessing the key or distinguishing outputs from random, which breaks critical security properties. This increases the adversary's advantage in distinguishing the PRF from random outputs, as it can potentially exploit this reduction in entropy.

4.3.2 Bounding the Adversary's Advantage

A key metric for evaluating the adversary's success in guessing the secret key k is derived from the min-entropy of the key's distribution. Specifically, the expression

$$2^{-H_\infty(k)}$$

quantifies the probability of an adversary successfully guessing the key in a single attempt, based on the likelihood of the most probable outcome in the distribution.

Formally, the adversary's success probability can be bounded below by the min-entropy of the key as follows:

If we aim to bound the adversary's advantage based solely on the min-entropy of the key, we can express it as:

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{A}}(f_k) \geq 2^{-H_\infty(k)}.$$

4.3.2.1 Ideal Case: Uniformly Random Keys

If we consider an ideal situation, where k is uniformly random and there are no exploitable structural patterns in the PRF, the adversary's advantage should be negligible. Since k is uniformly distributed over the entire key space $\mathbb{Z}_p$, then each key is equally likely. The probability of any particular value k_i is $\frac{1}{p}$. The min-entropy in this case is:

$$\mathrm{H}_\infty(k) = \log_2(p).$$

For

$$p = 2^\lambda$$

, this represents full entropy (e.g., λ-bit keys have λ bits of entropy) and ensures the key is uniformly distributed over an exponentially large key space.

However, when k is non-uniform and as the min-entropy decreases (as k becomes more predictable), $2^{-H_\infty(k)}$ increases, which directly increases the adversary's advantage as we can see by the bound expression. For instance, if the min-entropy of k is significantly lower than the maximum possible entropy (e.g, $H_\infty(k) = O(\log \lambda)$) the adversary can gain a clear statistical advantage in distinguishing the PRF outputs from random outputs since entropy scales logarithmically with λ instead of exponentially. This results in a polynomial key space making it computationally feasible for polynomial-time adversaries to have success with brute force attacks.

4.3.2.2 Limitations of Min-Entropy

This inequality provides a lower bound on the adversary's success probability, which reflects the fact that if the key distribution has low min-entropy, the adversary's advantage is at least as large as $2^{-H_\infty(k)}$ and if the key distribution has low min-entropy than this advantage is negligible, as shown in the previous subsection. However, in practice, the adversary's actual advantage could be higher since even if a key distribution has high min-entropy, it might have structural weaknesses. For example, subsets of keys could produce biased or predictable outputs for specific inputs that can allow the adversary to achieve a success probability significantly greater than $2^{-H_\infty(k)}$. Let us consider for example an extreme case where the PRF has an output length of 1 bit. We define the set BAD as follows:

$$\text{BAD} = \{k : F_k(0) = 0\}.$$

This means that BAD is the set of all secret keys k such that when applying the PRF F_k to the input 0, the output is 0. It is a large set, covering roughly half of the key space and even though keys sampled from this set have high min-entropy (since they are sampled uniformly from a large subset of the key space), the adversary can easily distinguish between the PRF and a random function.

In this case, despite the high min-entropy of the keys in BAD, the adversary can guess with high success probability,, approximately $\frac{3}{4}$, calculated as:

$$\frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{3}{4},$$

where:

- $\frac{1}{2}$ is the probability that the key is in BAD,
- $\frac{1}{2} \times \frac{1}{2}$ represents the probability that the key is not in BAD but still produces the output 0, since the PRF output is only 1 bit.

This example demonstrates that min-entropy alone does not fully capture the adversary's advantage. Even with a high min-entropy key, structural patterns in the PRF output can give the adversary a significant advantage.

4.3.2.3 Statistical Distance

To better understand and measure the quality of the key distribution beyond min-entropy, we can use the notion of statistical distance, defined as:

Statistical distance measures how close two distributions are. It is defined between two distributions P and Q over a set X as:

$$\Delta(P, Q) = \frac{1}{2} \sum_{x \in X} \left| \Pr[P = x] - \Pr[Q = x] \right|.$$

In the context of key distributions, P represents the actual key distribution $\mathcal{D}_\lambda$, and Q represents the uniform distribution $\mathcal{U}_\lambda$. The statistical distance $\Delta(\mathcal{D}_\lambda, \mathcal{U}_\lambda)$ tells us how much the key distribution deviates from the uniform distribution.

Here, P represents the actual key distribution $\mathcal{D}_\lambda$, and Q represents the uniform distribution $\mathcal{U}_\lambda$. For the PRF to remain secure, we require the statistical distance between $\mathcal{D}_\lambda$ and $\mathcal{U}_\lambda$ to be small. More formally, we will use the notation $\mathcal{D}_\lambda \approx_\epsilon \mathcal{U}_\lambda$ to denote that the statistical distance is bounded by ϵ, i.e., $\Delta(\mathcal{D}_\lambda, \mathcal{U}_\lambda) \leq \epsilon$.

By ensuring that $\Delta(\mathcal{D}_\lambda, \mathcal{U}_\lambda) \leq \epsilon$, we can argue that the PRF outputs are computationally indistinguishable from random, provided the adversary's distinguishing advantage is negligible.

4.3.3 Formal Proof of Concrete Security Impact with Low Min-Entropy

Let us formally prove that if the key k is sampled from a key distribution that has low min-entropy, there are concrete impacts in the security of the PRF $f_k(x) = H(x)^k$.

Theorem 4.3.1. Let $f_k(x) = H(x)^k$ be a PRF, where H is modeled as a hash function that maps its input to uniformly random elements from cyclic group $\mathbb{G}$ generated by g. If the key k is sampled with low min-entropy, $H_\infty(k) = O(\log \lambda)$ bits of min entropy, then there exists an efficient adversary $\mathcal{A}$ that can distinguish between $f_k(x)$ and a random function with non-negligible advantage.

Proof. We will prove the theorem by constructing an efficient adversary $\mathcal{A}$ that exploits the low min-entropy of the key k to mount a distinguishing attack against the PRF $f_k(x)$.

Step 1: Problem Setup:

The adversary $\mathcal{A}$ is given access to an oracle O, which either implements the PRF $f_k(x) = H(x)^k$ for some secret key k, or it implements a truly random function $R(x)$. The adversary's goal is to distinguish whether the oracle implements $f_k(x)$ or $R(x)$.

The key k is not chosen uniformly at random but instead from a distribution with min-entropy $H_\infty(k) = O(\log \lambda)$, meaning there are approximately λ^c possible values for k, where c is a constant.

Step 2: Adversary Strategy:

1. **Query the Oracle:** The adversary $\mathcal{A}$ chooses a fixed input x_1 and queries the oracle $O(x_1)$, obtaining either:
   - $O(x_1) = H(x_1)^k$ (if the oracle implements the PRF) or
   - $O(x_1) = R(x_1)$, where $R(x_1)$ is a random element from $\mathbb{G}$.
2. **Key Space Search:** Since k is drawn from a distribution with min-entropy $O(\log \lambda)$ this means that the number of required attempts grows polynomially, not exponentially, making practical attacks feasible. The adversary can perform an exhaustive

search over the λ^c possible values for k since the number of possible key values becomes polynomial in λ. For each candidate key k', the adversary computes:

$$H(x_1)^{k'} = g^{k'}$$

and compares it to the value $O(x_1)$.

3. **Distinguishing Oracle:**

   - If there exists a $k' \in \{0,1\}^{O(\log \lambda)}$ such that $H(x_1)^{k'} = O(x_1)$, the adversary concludes that the oracle is implementing the PRF $f_k(x) = H(x)^k$.
   - If no such k' is found, the adversary concludes that the oracle is implementing a truly random function $R(x)$.

Step 3: Analysis of Success Probability:

- **Case 1: Oracle Implements PRF.** In this case, there exists a correct key $k \in \{0,1\}^{O(\log \lambda)}$, and the adversary will find this key with probability 1 after searching through the λ^c possible keys.
- **Case 2: Oracle Implements Random Function.** In this case, the oracle's output is independent of any $H(x_1)^k$, and the adversary's exhaustive search will fail to find a matching key k', with high probability.

Thus, it becomes computationally feasible for the adversary to distinguish between the PRF and a random function.

Step 4: Concrete Security Impact:
When the key k is chosen from a distribution with low min-entropy (e.g., $H_\infty(k) = O(\log \lambda)$), meaning that entropy scales logarithmically instead of exponentially in λ, then the number of possible key values becomes polynomial in λ. Thus, the concrete security of the PRF diminishes enough that polynomial-time attacks, such as exhaustive key search, become practical. The security of the PRF $f_k(x) = H(x)^k$ is directly impacted by the low entropy of the key distribution, since it becomes computationally feasible for the adversary to distinguish whether the oracle implements $f_k(x)$ or $R(x)$, thus breaking the pseudorandomness of the PRF.

□

4.3.4 Impact on Security Properties

After this analysis we get to the conclusion that when the key k is non-uniform, several critical security properties of the 2HashDH protocol and the underlying PRF are compromised:

1. **Pseudorandomness**: The pseudorandomness of $f_k(x) = H(x)^k$ relies on the key k having high entropy. A non-uniform key with low min-entropy means the outputs $H(x)^k$ are more predictable, allowing an adversary to distinguish them from random with non-negligible advantage.

2. **Indistinguishability**: The ability of an adversary to distinguish the PRF from a random function is directly tied to the uniformity of the key. If the key is non-uniform, the indistinguishability of the PRF is no longer guaranteed since he adversary can exploit the predictable key distribution to increase their advantage in distinguishing the PRF from a truly random function.

3. **Protocol Security**: In the context of the 2HashDH VOPRF protocol, the security of the protocol heavily relies on the pseudorandomness and unpredictability of the underlying PRF $H(x)^k$. The assumption is that the secret key k is uniformly distributed across the key space, ensuring that the PRF behaves unpredictably and resists attacks from adversaries. However, the potential pitfall arises when the server, whether intentionally or unintentionally, selects a non-uniform key. The client, which relies on the server to perform computations correctly, verifies the correctness of the computations via public key-based checks (such as the DDH check or the NIZK proof in the VOPRF). These checks verify that the computation is consistent with the public key $y = g^k$ but do not provide any guarantees about the distribution of the secret key k used in the $H(x)^k$ computation, this means that the client might unknowingly have a weak key used in the $H(x)^k$ computation, assuming full security guarantees, when in fact the key is predictable and exploitable. This means that if the key used in the protocol has low min-entropy, the protocol's guarantees of secrecy, authenticity, and overall security are undermined.

In conclusion, the security of the 2HashDH VOPRF protocol and its underlying PRF $f_k(x) = H(x)^k$ critically depends on the uniformity and high min-entropy of the key k. When the key is non-uniform or has low min-entropy, the adversary's advantage increases, leading to a breakdown in the pseudorandomness and indistinguishability properties that are essential for the protocol's security. It is therefore imperative to ensure that the key k is chosen uniformly at random to maintain the integrity and security of the protocol.

5

Mitigating Predictable Keys in the 2HashDH OPRF

In this chapter we investigate if it is possible to mitigate the risks posed by a weak server-side key being used in an OPRF protocol. Specifically, our objective is to explore whether the client-side entropy can compensate for the lack of entropy in the key, thereby maintaining the pseudorandomness of the overall construction. To do this we will be exploring the concept of dual PRFs.

5.1 The Problem and our Approach

The current approach of OPRF protocols places the client's trust heavily in the server's key rotation process, where the key is assumed to be uniformly random. The reason for this is that in the real world OPRFs are run by servers that hold a cryptographic keypair, where the private key is used for the PRF computation and the public key is used to provide clients with the ability to derive trust in the final output that they receive (as we explained in the VOPRF functionality). As with all cryptographic protocols, to avoid security breaches for any protocol participants in the case of a key compromise event, it is essential that the keypair that is in use is rotated frequently. These key rotations involve the OPRF server internally deriving a new keypair randomly and revealing the new public part of the keypair to prospective clients. However, if a server becomes corrupted (intentionally or not) and can no longer derive truly random keypairs, this server key rotation can lead to non-uniform keys being used in the protocol and as we demonstrated in the previous chapter in our analysis of the 2HashDH OPRF construction, non-uniformity of the server-side key can significantly degrade the security of the protocol, particularly in terms of pseudorandomness.

In this chapter, we propose a mitigation strategy that addresses the risks associated with weak server-side keys by exploring whether it is possible to derive additional randomness from the client-side input, thereby avoiding sole reliance on the server's key. By potentially extracting pseudorandomness from both the key and the input, we aim to strengthen the security of the OPRF even when the server's key is weak.

This mitigation approach is relevant because it seeks to reduce the trust placed exclusively in the server by allowing the client to play a more active role in the security of the protocol. This configuration would provide stronger trust guarantees for clients, enabling them to rely not only on the server's key rotation process but also on their own input to secure the protocol. The relevance of this approach is grounded in the vulnerabilities identified in Chapter 4, where we proved that the non-uniformity of k leads to a loss of security. This configuration would lead to more transparent key rotation strategies that allow clients to derive stronger trust guarantees from the entire exchange.

Proceeding in this chapter, we will:

- **Leverage the dual PRF concept** to analyze whether pseudorandomness can be derived from the input x to compensate for the lack of entropy in the key k. This approach allows us to explore the input's role in maintaining security.
- **Examine under what assumption $H(x)^k$ remains a PRF when "swapped"**, specifically focusing on scenarios where x is chosen uniformly and k is potentially weak or adversarially chosen.
- **Prove that $H(x)^k$ satisfies the dual PRF definition**, this will help us explore whether entropy can be derived from both the key and the input source.
- **Propose a dual PRF configuration for $H(x)^k$ and develop a security model** in which entropy is derived from both the key k and the input x, ensuring that it fits the properties of a dual PRF while accounting for the interaction between key and input entropy, providing robust security guarantees in scenarios where both the key and input play a role in maintaining pseudorandomness.

5.2 $H(x)^k$ as a Dual PRF

In Subsection 2.3.3, we gave a brief introduction to the concept of dual PRFs, explaining that a function family is considered a dual PRF is a PRF when keyed normally and also when the roles of its key and input are swapped. This concept is relevant to us, as it provides a framework in which we can study whether input x can contribute to the pseudorandomness of $H(x)^k$. It is particularly relevant for the analysis in this chapter because it allows us to investigate whether both the server-side key k and the client input x can contribute to the pseudorandomness of the function.

The importance of the dual PRF lies in its ability to ensure security even when the conventional roles of key and input are swapped, which aligns with our goal of analyzing whether the input can play a more active role in maintaining security.

To begin this analysis, we will take into account the formal definition of a dual PRF that we presented in Section 2.3.3 and we will formulate under what assumption is $H(x)^k$ a PRF when x is uniform and k is chosen by the adversary. This will allow us to formally

assess whether $H(x)^k$ satisfies the dual PRF definition, which will help us conclude if it is possible to mitigate the use of a weak server side key in the protocol if the client uses a a strong input, chosen uniformly from a high-entropy distribution (e.g., a strong password).

5.2.1 Formal Analysis of $H(x)^k$ when x is Uniform and k is Chosen by the Adversary

In this section, we conduct a formal analysis of the PRF $H(x)^k$ under the swapped configuration, where the input x is uniformly chosen from a high-entropy distribution, and the key k is adversarially chosen. We aim to explore under which assumptions $H(x)^k$ remains secure as a PRF in this setting.

5.2.1.1 Configuration Setup

Let:

- G be a cyclic group of prime order p, generated by an element g.
- $H : [0,1]^\lambda \to G$ be a hash function modeled as a random oracle. The function $H(x)$ produces uniformly random elements from a group $\langle g \rangle$.
- $x \xleftarrow{\$} \mathcal{D}_\lambda$ this means that x is sampled uniformly at random from a high-entropy distribution $\mathcal{D}_\lambda$ where $H_\infty(x) \geq \lambda$.
- k is adversarially chosen, meaning the adversary has complete control over k.

The function we are analyzing is defined as:

$$H(x)^k = g^{ak}$$

where $g^a = H(x)$ is derived from the uniformly random input x.

The goal is to determine whether $H(x)^k$ behaves as a pseudorandom function in this configuration and we can already understand that for this to work we have to make the assumption that the adversary cannot know the value of x since if the adversary knew x it would be trivial to calculate $H(x) = g^a$ and then, since the adversary is in control of k in this configuration, he would be able to easily compute $H(x)^k = g^{ak}$ and distinguish it from a random group element, thus breaking the PRF security. This assumption of the adversary not knowing the value of x ensures that the adversary cannot compute $g^a = H(x)$ directly, which is key to maintaining the pseudorandomness of $H(x)^k$.

Theorem 5.2.1. Let $f_k(x) = H(x)^k$ be a PRF, where H is modeled as a hash function that maps its input to uniformly random elements from cyclic group $\mathbb{G}$ generated by g, and where x is uniformly random and k is chosen by the adversary, the adversary does not know x. If there exists a polynomial-time adversary $\mathcal{A}$ that distinguishes between $f_k(x)$ and a truly random function with non-negligible advantage, then we can construct a

polynomial-time algorithm $\mathcal{B}$ that solves the Decisional Diffie-Hellman (DDH) problem with non-negligible probability.

Proof. We prove the theorem above by showing that if there exists an adversary $\mathcal{A}$ that can distinguish $H(x)^k$ from a random function with non-negligible advantage, we can construct a reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the DDH assumption. This shows that the PRF security of $H(x)^k$ holds as long as the DDH assumption is valid.

In the DDH problem, the adversary is given a tuple (g, g^a, g^k, T), where:

- $g^a \leftarrow H(x)$ is derived from a uniformly random x,
- g^k is chosen by the adversary,
- T is either g^{ak} or a random element g^r from G.

The adversary must distinguish whether $T = g^{ak}$ or $T = g^r$. If the adversary can distinguish this with non-negligible probability, it breaks the DDH assumption.

5.2.2 Construction of Reduction

We construct a reduction $\mathcal{B}$ that uses the adversary $\mathcal{A}$ to break the DDH assumption. We do this by simulating the PRF game for $\mathcal{A}$, using the DDH challenge tuple (g, g^a, g^k, T) as input to $\mathcal{B}$.

Input to $\mathcal{B}$: $\mathcal{B}$ receives a DDH challenge tuple (g, g^a, g^k, T), where T is either g^{ak} or a random element g^r. The goal of $\mathcal{B}$ is to determine whether T is g^{ak} or g^r using the adversary $\mathcal{A}$.

Simulation of the PRF Game: $\mathcal{B}$ interacts with $\mathcal{A}$ by simulating the PRF game as follows:

- $\mathcal{B}$ sets $g^a \leftarrow H(x)$, where x is sampled uniformly at random from a high-entropy distribution, meaning x is unknown to $\mathcal{A}$.
- $\mathcal{B}$ sets g^k according to $\mathcal{A}$'s choice of k. This simulates the adversarial control of k.
- For each query made by $\mathcal{A}$, $\mathcal{B}$ returns the value of T:
  - If $T = g^{ak}$, $\mathcal{B}$ simulates the real PRF by returning g^{ak} as the response to $\mathcal{A}$'s query.
  - If $T = g^r$, $\mathcal{B}$ simulates the random function by returning g^r as the response to $\mathcal{A}$'s query.

Adversary's Success: If $\mathcal{A}$ can distinguish between the real PRF $H(x)^k$ and a random function with non-negligible advantage, then it implies that $\mathcal{B}$ can distinguish whether $T = g^{ak}$ or $T = g^r$. Hence, $\mathcal{B}$ breaks the DDH assumption by distinguishing g^{ak} from a random group element.

Since an adversary $\mathcal{A}$ breaking the PRF security of $H(x)^k$ directly leads to breaking the DDH assumption, we conclude that $H(x)^k$ is a secure pseudorandom function under the DDH assumption, as long as x is uniformly random and the adversary does not know the value of x. □

5.2.3 Proving $H(x)^k$ as a Dual PRF

We have now established that $H(x)^k$ is a PRF when keyed by the input x. This proves that $H(x)^k$ is a swap-PRF, as by definition, a swap-PRF is a PRF when the roles of the key and input are switched. Specifically, we have shown that when x is used as the key and k is treated as the input, the function still behaves pseudorandomly under the assumptions that $H(x)$ acts like a random oracle, the adversary does not know the value of x, and that the security is based on the DDH assumption.

In Chapter 4, we also formally proved that $H(x)^k$ is a PRF when used in the traditional manner, that is when k is the key and x is the input. We did this by proving theorem 4.2.2.

5.2.3.1 Dual PRF Definition

Following the formal definiton of a dual PRF given in Subsection 2.3.3, a dual PRF is a function that satisfies two conditions:

1. It is a PRF when keyed by the first input.
2. It is a swap-PRF, meaning it remains a PRF when the key and input roles are reversed (when keyed by the second input).

Since we have now proven that $H(x)^k$ satisfies both of these conditions:

- In its standard form, $H(x)^k$ is a PRF when keyed by k (k being uniform)with input x (as proven in theorem 4.2.2).
- In its swapped form, $H(x)^k$ remains a PRF when keyed by x (x being uniform) with input k, as shown above.

This formally establishes that $H(x)^k$ is a dual PRF, under the assumption that $H(x)$ behaves like a random oracle, x is uniform and the adversary does not know x. The function retains its pseudorandomness whether keyed by the first or second input, fulfilling the definition of a dual PRF. Relying on the assumption that the key (whether it's k or x) is sampled uniformly.

5.3 Security Model for the Dual PRF Configuration

Now that we understand under what assumptions $H(x)^k$ is a PRF, when x is uniform and k is chosen by the adversary we can define our security model for the dual PRF construction.

Our model is designed with the aim of enhancing the functionality of OPRFs, deriving entropy from both the key and the input by leveraging the benefits of a dual PRF being a PRF not only when keyed conventionally (through its first input), but also when "swapped" and keyed (unconventionally) through its second (message) input.

In Chapter 3 we defined the Standard PRF security model, in this section we will modify that definition with the aim of expressing the standard PRF model and dual PRF model with our security model. In particular, we will modify the definition so that the adversary specifies the input distribution from which the input will be sampled, rather than the input itself. We do this because of the conclusions of the last section were we found that $H(x)^k$ is a PRF if swapped (meaning when x is uniform and k is chosen by the adversary) under the assumptions that x is uniformly random and that the adversary does not know the value of x.

In this initial section of the security model, we will formally define the key and input sampling for the PRF and present a formal definition of the PRF.

We do this by modifying the standard definition so that the adversary specifies the input distribution rather than the input. This way we can align our model with the definition of a dual PRF, that is, a PRF that is a PRF when keyed as usual by the first input, but also if keyed by the second input. We will dive deeper into this topic in this chapter where we present our proposed dual PRF model where entropy is derived from both the key and the input source.

5.3.1 Key Sampling

Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of distributions parameterized by a security parameter λ.

The key k is then sampled according to the distribution $\mathcal{D}_\lambda$:

$$k \leftarrow \mathcal{D}_\lambda$$

5.3.2 Input Sampling

Let $\mathcal{E} = \{\mathcal{E}_\lambda\}_{\lambda \in \mathbb{N}}$ be another family of distributions parameterized by a security parameter λ.

For each new query that the adversary makes, the input x is sampled from a family of distributions $\mathcal{E}$

$$\text{For each new query that the adversary makes, } x \leftarrow \mathcal{E}_\lambda.$$

We model the key and input sampling like this since in our configuration the pseudo-randomness of the function is derived from both the key k and the input x, each sampled from their respective distributions.

The key k is sampled from the family of distributions $\mathcal{D}_\lambda$, while the input x is sampled from the family of distributions $\mathcal{E}_\lambda$. The min-entropy of these distributions is critical to ensure the unpredictability and security of the system.

- The min-entropy of the key k is determined by the min-entropy of the distribution $\mathcal{D}_\lambda$, while the min-entropy of the input x is determined by the min-entropy of the distribution $\mathcal{E}_\lambda$.
- Specifically, the min-entropy of the key k is defined as $H_\infty(k) = H_\infty(\mathcal{D}_\lambda)$, where $H_\infty(\mathcal{D}_\lambda)$ is the min-entropy of the distribution $\mathcal{D}_\lambda$.
- The min-entropy of the input x is defined as $H_\infty(x) = H_\infty(\mathcal{E}_\lambda)$, where $H_\infty(\mathcal{E}_\lambda)$ is the min-entropy of the distribution $\mathcal{E}_\lambda$.

5.3.3 Libraries

We will also define the libraries like we defined in Chapter 3:

- $\mathcal{L}^F_{\text{dualPRF-real}}$ represents a dual PRF.
- $\mathcal{L}^F_{\text{dualPRF-rand}}$ represents a function chosen uniformly at random from the set of all functions having the same domain and range.

$\mathcal{L}^F_{\textbf{dualPRF-rand}}$

T := empty associative array

for every new query that the adversary makes: $x \leftarrow \mathcal{E}_\lambda$

lookup(x):

---

if $T[x]$ is undefined:
  $T[x] \leftarrow \{0,1\}^\lambda$
return $T[x]$

$\mathcal{L}^F_{\textbf{dualPRF-real}}$

$k \leftarrow \mathcal{D}_\lambda$

for every new query that the adversary makes:
$x \leftarrow \mathcal{E}_\lambda$

lookup(x):

---

return $F(k, x)$

Definition 11 (Dual PRF Security)**.** *Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a deterministic function.*

We say that F is $\mathcal{D}$ an $\mathcal{E}$-secure if $\mathcal{L}^F_{dualPRF\text{-}real} \approx \mathcal{L}^F_{dualPRF\text{-}rand}$ when F is keyed conventually by k with input x but also when keyed by x in a swapped configuration, where $\approx$ denotes computational

indistinguishability, meaning that no efficient (polynomial-time) algorithm can distinguish between the two distributions with a non-negligible probability.

5.3.4 Security Through Entropy Extraction in the Dual PRF $H(x)^k$

As demonstrated in the previous sections, the pseudorandomness of the dual PRF $H(x)^k$ is derived from the uniformity of the key input (whether it's k or x). This means that if k is non-uniform we could maintain the security of the function if we keyed it with an uniform x.

However, we are also interested in a configuration were the pseudorandomness is derived from the entropy of both the key k and the input x, as our goal is to investigate the combined impact of their entropy on the overall security guarantees. We are particularly interested in the scenario where the server is compromised or corrupted, generating a key k with low min-entropy. In such cases, we aim to mitigate the weakness of the key by extracting sufficient entropy from the strong, high-entropy client input x. By leveraging entropy from both the key and the input, we ensure that the PRF remains secure even in the presence of a compromised server that can no longer derive uniform keys.

To formally understand how this construction would be able to retain security under such conditions, we will leverage the concept of randomness extractors.

5.3.4.1 Randomness Extractors

A randomness extractor [34] is a deterministic algorithm that produces nearly uniform bits given access to a weak random source. The quality of the output is measured using the statistical distance between two distributions, we defined statistical distance in Subsection 4.3.2.3.

A randomness extractor for a family of distributions can be formally defined as follows:

Definition 12 ([34]). *Let X be a family of distributions on the universe $\{0,1\}^n$. A function $Ext : \{0,1\}^n \to \{0,1\}^m$ is called an ε-extractor for X if for any distribution $X \in \mathcal{X}$, we have:*

$$\Delta(Ext(X); U_m) \leq \varepsilon.$$

The parameter ε is called the error of the extractor.

5.3.4.2 Two-Source Extractors

Given that our goal is to combine entropy from both the key k and the input x, we make use of a two-source extractor. A two-source extractor is designed to extract uniform randomness from two weak but independent sources.

Definition 13 ([83]). *A function $TExt : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ is a strong two-source extractor for min-entropy k_1, k_2 and error ε if for every independent (n_1, k_1) source X and (n_2, k_2) source Y, we have:*

$$|(TExt(X,Y), X) - (U_m, X)| < \varepsilon$$

and

$$|(TExt(X,Y),Y) - (U_m,Y)| < \varepsilon,$$

where U_m is the uniform distribution on m bits independent of (X,Y).

As pointed out by Li in [82], explicit constructions of such functions have been highly challenging. However these constructions have been proved to exist, with the first explicit construction of a two-source extractor presented by Chor and Goldreich [37]. In this thesis we are more interested in Raz's work [100], which provided a two source extractor that requires one source to have min-entropy larger than $\lambda/2$, while the other source can have min-entropy as low as $O(\log \lambda)$.

This implies that if the min-entropy of k and x are both large enough $H(x)^k$ is indeed a secure PRF in the setting of a strong two-source extractor. Moreover, it shows that even if one source has low min-entropy it is possible to mitigate it by having the other source be sampled from high min-entropy distribution.

5.3.4.3 Implications

In this configuration of the dual PRF $H(x)^k$, the pseudorandomness depends on the combined min-entropy of both the key k and the input x, which are sampled from their respective distributions. In the case of the server being corrupted and the key k being sampled from a distribution with low min-entropy (as low as $H_\infty(k) = O(\log \lambda)$), we need to ensure that the input x, provided by the client, has sufficient entropy to maintain security ($H_\infty(x) \geq \lambda/2$).

Additionally, it's important to ensure that the statistical distance between the output distribution and uniform distribution remains small, and the outputs remain computationally indistinguishable from random functions to maintain security.

This can be very valuable for the OPRF construction that we analyzed in Chapter 4, and for other OPRFs, since the client can derive stronger trust guarantees, from the entire exchange by using a strong input. As a strong input can compensate for a low-entropy server-side key, the client is not required to fully rely on the server's key rotation and key generation.

6

Implications for deployed OPRF Protocols

In this chapter, we discuss the broader implications of the results obtained in this thesis, particularly focusing on the potential vulnerabilities in deployed OPRF protocols when the server key is non-uniform or corrupted, and how our proposed mitigations can be applied.

6.1 Impact of Non-Uniform Server Keys on OPRF Protocols

One of the central results of this thesis is the demonstration of a concrete security impact when the OPRF protocol's server key is non-uniform. Specifically, in Chapter 4, we analyzed and proved that when a server is either intentionally or unintentionally corrupted and the key rotation process is no longer able to derive uniform keys, the security guarantees of the OPRF protocol may degrade significantly.

This loss of security arises because the strength of the OPRF protocol hinges on the uniformity of the server-side key k. When k is no longer sampled uniformly, it can result in the output of the protocol becoming predictable, exposing the protocol to cryptographic vulnerabilities. A non-uniform key k essentially undermines the pseudorandomness that the protocol relies upon, allowing adversaries to mount attacks by exploiting this predictability.

In practical terms, this is particularly concerning for real-world OPRF deployments in cloud-based or privacy-preserving systems, where the server plays a pivotal role in generating secure keys. If a server is compromised or misconfigured, clients unknowingly rely on weak keys, and the entire system's security may collapse. The findings in this thesis demonstrate the critical importance of ensuring that key rotation mechanisms remain robust and continue to derive truly random keys, even in the face of corruption or adversarial interference.

6.2 Vulnerabilities in VOPRF Protocols

This vulnerability becomes even more pressing in VOPRFs protocols. In VOPRFs, the client performs checks to ensure that the server is behaving honestly. However, as we have shown in our investigation, when the server key is non-uniform, all verifiable checks from the client's perspective may still pass, despite the fact that the server is using a weak key.

This creates a dangerous scenario where the client, after performing all verification steps, might proceed with the protocol under the false assumption that security guarantees are intact. As a result, the client could, for instance, encrypt their sensitive data using a weak key, exposing it to future attacks. The ability of an adversary to manipulate key rotation or the key generation process effectively invalidates the security model of the VOPRF, leaving the client and system vulnerable without detection.

The practical consequence here is that VOPRF protocols must be designed with an awareness of this potential vulnerability, and extra safeguards must be implemented to ensure that key rotation processes cannot be silently compromised. Strengthening the verification process, or introducing transparency into the key rotation process, may be necessary to mitigate these risks.

6.3 Mitigation Through Input-Derived Entropy: Dual PRFs and Transparent Key Rotation

In Chapter 5, we explored an alternative approach to mitigate the vulnerabilities caused by non-uniform server keys. Specifically, we proved that $H(x)^k$, a central PRF construction in OPRF protocols, can potentially function as a dual PRF. A dual PRF retains pseudorandomness even when the roles of key and input are swapped. This allowed us to propose a configuration where entropy is derived not just from the server-side key k, but also from the client-side input x.

By introducing the ability to derive entropy from both the key and the input, we mitigate the reliance on the server key alone. In scenarios where the server key is weak or non-uniform, a strong, high-entropy client input can still ensure that the output of the OPRF protocol retains pseudorandomness. This approach provides an additional layer of security, especially in situations where server-side processes (like key rotation) cannot be fully trusted.

Furthermore, this construction opens up the possibility of exploring more transparent key rotation strategies. In traditional OPRF protocols, the client places significant trust in the server's ability to generate and rotate secure keys. By allowing the client to play a more active role in the entropy generation process, we reduce the need for blind trust in the server's key management. This shift towards more transparent key rotation mechanisms enables clients to derive stronger trust guarantees from the entire OPRF exchange.

6.4 Real-World Applications: OPAQUE and Privacy Pass

The results of this thesis have immediate implications for several real-world cryptographic protocols that rely on OPRFs for security. Two key examples of such protocols are OPAQUE and Privacy Pass. Below, we discuss how the vulnerabilities we uncovered, as well as the proposed mitigations, affect these protocols in practice.

6.4.1 OPAQUE: Password-Authenticated Key Exchange (PAKE)

OPAQUE is a widely discussed password-authenticated key exchange protocol, which leverages an OPRF to securely store and retrieve password-based secrets. In OPAQUE, the user provides a password, and the server stores a corresponding OPRF output that it uses to verify the user's authentication attempts. The protocol provides strong security guarantees by preventing the server from learning the user's password directly.

6.4.1.1 Implications for OPAQUE:

The security of OPAQUE relies on the assumption that the server generates and rotates keys uniformly at random, ensuring that the OPRF output remains pseudorandom and secure. However, as demonstrated in this thesis, if the server is corrupted and the key rotation process produces non-uniform keys, this directly compromises the security of OPAQUE.

- *Loss of Security:* If the server is compromised and cannot generate uniform keys, the OPRF output used for password verification may become predictable. This makes it easier for an adversary to mount offline dictionary attacks by leveraging weak or non-uniform keys to brute-force the user's password.

- *Client Trust:* Since OPAQUE's security is based on an OPRF output, the client unknowingly trusts the server to generate secure keys. If key rotation is faulty, the user's password-based secrets are exposed to attacks, undermining the core security guarantees of the protocol.

6.4.1.2 Mitigation:

The mitigation strategies proposed in Chapter 5, deriving entropy from both the key and the input, can be applied to OPAQUE to reduce reliance on the server's key management process. By deriving entropy from both the user's password (the client input) and the server-side key, OPAQUE can preserve its security guarantees, even if the server generates weak keys. This provides an extra layer of security that compensates for potentially compromised server key rotation processes.

6.4.2 Privacy Pass: Anonymous Authentication with OPRFs

Privacy Pass is a protocol that allows users to anonymously authenticate with services without revealing their identity. It uses OPRFs to generate tokens that users can later redeem to prove they are legitimate users, while preserving their privacy.

6.4.2.1 Implications for Privacy Pass:

Privacy Pass heavily relies on the server to generate OPRF tokens securely. The tokens are produced by combining the user's input (a random value) with the server's OPRF key. In this setup, the server's OPRF key is crucial for ensuring that the tokens are pseudorandom and unlinkable, preserving the user's anonymity.

However, as we demonstrated in this thesis, if the server's key becomes predictable or non-uniform due to a corruption or faulty key rotation process, the entire system breaks down:

- *Loss of Anonymity:* An attacker who compromises the server's key can link token requests back to the users, breaking the anonymity guarantees that Privacy Pass is designed to provide.
- *Predictable Tokens:* The predictability of the OPRF tokens allows an attacker to predict valid tokens or replay them, violating the security and privacy guarantees of the protocol.

6.4.2.2 Mitigation:

By applying the results from Chapter 5, Privacy Pass can be improved to derive security from both the client-side input and the server's key. The client's input, used to generate the OPRF tokens, can provide enough entropy to preserve pseudorandomness, even when the server's key is non-uniform or weak. This makes the system more robust against server-side corruption, as the client input can compensate for any vulnerabilities in the key generation process.

6.5 Key Well-Formedness and Protocol Limitations

It is important to note that in certain OPRF protocols, deriving randomness from client input may not be feasible due to the nature of the input. For instance, in privacy-preserving systems like PSI or certain anonymous authentication schemes, the client's input may be fixed, highly structured, or inherently low in entropy. In such cases, solely relying on client input to bolster security may not be possible. These protocols often depend on the server's ability to generate strong keys, making server-side key rotation mechanisms critical for maintaining the security of the entire protocol.

In scenarios where input-derived entropy is impractical, focusing on ensuring server key well-formedness becomes even more important. One avenue for improving the

security of OPRF protocols is by incorporating key well-formedness checks. Ensuring that keys adhere to well defined security properties, such as being uniformly random and meeting specific entropy requirements, can significantly bolster security guarantees. As noted by Casacuberta et al. [32], the concept of key well-formedness has not been fully explored in the context of OPRF protocols but presents a valuable direction for enhancing both security and resilience against key-related attacks.

Techniques such as verifiable random functions (VRFs) or publicly verifiable randomness can help ensure that the key generation process remains robust and transparent, even in situations where the client cannot contribute significant entropy.

7

Conclusion and Future Work

7.1 Main Contributions

This thesis has explored the critical role of keys in the security of OPRF protocols. In particular, we have shown the vulnerabilities that can arise when servers generate non-uniform keys, either intentionally or due to server corruption, and how weak key generation undermines the security guarantees of OPRFs.

A significant contribution of this thesis is the formal analysis of weak keys sampled from non-uniform distributions and their concrete security implications for an OPRF. We formally demonstrated that non-uniformity in server-generated keys introduces severe vulnerabilities, making traditional key rotation mechanisms insufficient. This highlights the necessity for high-entropy, unpredictable key generation, especially in scenarios where servers may be compromised.

We also proposed an alternative solution leveraging dual PRFs, combining entropy from both the key and the client input. This approach mitigates the reliance on the server's key rotation process and enhances security by allowing clients to select a strong input to compensate for a weak or non-uniform server-generated key, allowing clients to derive stronger trust guarantees. Our formal analysis showed that this dual PRF construction remains secure under certain assumptions, even when the server-generated key has low min-entropy.

Moreover, this thesis contributes to the broader cryptographic discourse on OPRFs, particularly in their critical role as tools for privacy protection. By addressing the challenges of secure key generation and rotation, especially in real-world applications such as OPAQUE and Privacy Pass, we further demonstrated how OPRFs can enhance both privacy and security. Our work introduces novel approaches to mitigate weak key vulnerabilities, ensuring the continued efficacy of OPRFs in safeguarding sensitive data across various privacy-preserving protocols.

7.2 Future Work

While this thesis has contributed to the investigation of OPRF cryptography, several open questions and potential research directions remain.

An interesting direction would be to integrate well-formedness of keys into OPRF protocols. Incorporating this aspect could significantly strengthen the security guarantees of OPRF-based applications. By ensuring that the generated keys adhere to a well-defined structure, we could prevent a range of vulnerabilities caused by poorly formed keys or malicious key generation. This would add another layer of robustness to the overall cryptographic guarantees provided by OPRFs in practice.

Finding efficient post-quantum secure implementations of OPRFs is an area of active research [3, 19]. Given the looming threat posed by quantum computing, which could break many classical cryptosystems, it's crucial to explore quantum-resistant cryptographic primitives. While this thesis focuses on classical key rotation strategies and entropy-based security, the insights gained here may help inform the development of post-quantum OPRF protocols that combine secure key generation and input randomness. Investigating whether dual PRFs can be adapted to post-quantum cryptographic models is a natural extension of this work.

Finally, practical implementation and evaluation of the proposed dual PRF configuration would be needed for real-world adoption. Future research could focus on deploying this construction in existing OPRF-based protocols (e.g, OPAQUE) and evaluating its performance and scalability. Investigating its computational overhead and feasibility in large-scale systems would be critical for practical deployment.

Bibliography

[1] M. Abdalla et al. *Robust Password-Protected Secret Sharing*. Cryptology ePrint Archive, Paper 2016/123. 2016. DOI: 10.1007/978-3-319-45741-34. URL: https://eprint.iacr.org/2016/123 (cit. on p. 34).

[2] D. Aggarwal et al. *On Secret Sharing, Randomness, and Random-less Reductions for Secret Sharing*. Cryptology ePrint Archive, Paper 2021/802. 2021. URL: https://eprint.iacr.org/2021/802 (cit. on p. 19).

[3] M. R. Albrecht et al. "Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices". In: *Public-Key Cryptography – PKC 2021*. Ed. by J. A. Garay. Cham: Springer International Publishing, 2021, pp. 261–289. ISBN: 978-3-030-75248-4 (cit. on pp. 1, 33, 79).

[4] J. Alwen, Y. Dodis, and D. Wichs. *Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model*. Cryptology ePrint Archive, Paper 2009/160. 2009. URL: https://eprint.iacr.org/2009/160 (cit. on p. 19).

[5] Y. Angel et al. "Post-Quantum Noise". In: *ACM CCS 2022*. Ed. by H. Yin et al. ACM Press, 2022-11, pp. 97–109 (cit. on p. 25).

[6] M. Backendal et al. "When Messages Are Keys: Is HMAC a Dual-PRF?" In: *Advances in Cryptology – CRYPTO 2023*. Ed. by H. Handschuh and A. Lysyanskaya. Cham: Springer Nature Switzerland, 2023, pp. 661–693 (cit. on p. 25).

[7] A. Bagherzandi et al. "Password Protected Secret Sharing". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. 2011 (cit. on pp. 7, 34, 49).

[8] C. Baum et al. "PESTO: Proactively Secure Distributed Single Sign-On, or How to Trust a Hacked Server". In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2020 (cit. on pp. 7, 34, 49).

[9] M. Bellare et al. "The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme". In: *Journal of Cryptology* (2003) (cit. on p. 53).

[10] M. Bellare. "New proofs for NMAC and HMAC: Security without collision resistance". In: *Journal of Cryptology* 28.4 (2015-10), pp. 844–878 (cit. on p. 25).

[11] M. Bellare. "New proofs for NMAC and HMAC: Security without collision-resistance". In: *CRYPTO 2006*. Ed. by C. Dwork. Vol. 4117. Lecture Notes in Computer Science (LNCS). Heidelberg: Springer, 2006-08, pp. 602–619 (cit. on p. 25).

[12] M. Bellare, R. Canetti, and H. Krawczyk. "Keying Hash Functions for Message Authentication". In: *Advances in Cryptology – CRYPTO'96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Heidelberg: Springer, 1996-08, pp. 1–15 (cit. on p. 25).

[13] M. Bellare and A. Lysyanskaya. "Symmetric and Dual PRFs from Standard Assumptions: A Generic Validation of a Prevailing Assumption". In: *Journal of Cryptology* 37 (2024), pp. 33–54. DOI: 10.1007/s00145-024-09513-6. URL: https://doi.org/10.1007/s00145-024-09513-6 (cit. on p. 25).

[14] S. Bellovin and M. Merritt. "Encrypted key exchange: password-based protocols secure against dictionary attacks". In: *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*. 1992, pp. 72–84. DOI: 10.1109/RISP.1992.213269. URL: https://www.cs.columbia.edu/~smb/papers/neke.pdf (cit. on p. 35).

[15] N. Bindel et al. "Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by J. Ding and R. Steinwandt. Heidelberg: Springer, 2019, pp. 206–226 (cit. on p. 25).

[16] S. R. Blackburn, C. Cid, and C. Mullan. "Group Theory in Cryptography". In: *Department of Mathematics, Royal Holloway, University of London* (2010) (cit. on p. 26).

[17] D. Boneh. "The Decision Diffie-Hellman Problem". In: *Proceedings of the Third International Symposium on Algorithmic Number Theory (ANTS-III)*. Vol. 1423. Lecture Notes in Computer Science (LNCS). Springer, 1998, pp. 48–63. DOI: 10.1007/BFb0054851. URL: https://crypto.stanford.edu/~dabo/papers/DDH.pdf (cit. on p. 29).

[18] D. Boneh, D. Kogan, and K. Woo. "A Post-Quantum Round-Optimal Oblivious PRF from Isogenies". In: *ASIACRYPT 2020*. Vol. 12492. LNCS. Springer, 2020, pp. 520–550. URL: https://doi.org/10.1007/978-3-030-64834-3_18 (cit. on p. 1).

[19] D. Boneh, D. Kogan, and K. Woo. "Oblivious Pseudorandom Functions from Isogenies". In: *Advances in Cryptology – ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Cham: Springer International Publishing, 2020, pp. 520–550 (cit. on pp. 33, 34, 79).

[20] C. Bosley and Y. Dodis. "Does Privacy Require True Randomness?" In: *Theory of Cryptography*. Ed. by S. P. Vadhan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–20 (cit. on p. 19).

[21] D. Bourdrez et al. *The OPAQUE Asymmetric PAKE Protocol*. Internet-Draft draft-irtf-cfrg-opaque-11. Work in Progress. Internet Engineering Task Force, 2023. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/11/ (cit. on pp. 7, 9, 36–38).

[22] T. Bradley. *OPAQUE: The Best Passwords Never Leave your Device*. Cloudflare Blog. 2020. URL: https://blog.cloudflare.com/opaque-oblivious-passwords/ (visited on 2023-05-24) (cit. on pp. 39, 40).

[23] C. Brzuska, E. Cornelissen, and K. Kohbrok. "Security Analysis of the MLS Key Derivation". In: *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2022-05, pp. 2535–2553 (cit. on p. 25).

[24] C. Brzuska et al. "Key-Schedule Security for the TLS 1.3 Standard". In: *ASIACRYPT 2022, Part I*. Ed. by S. Agrawal and D. Lin. Vol. 13791. Lecture Notes in Computer Science (LNCS). Heidelberg: Springer, 2022-12, pp. 621–650 (cit. on pp. 18, 25).

[25] J. Camenisch et al. "Memento: How to Reconstruct Your Secrets from a Single Password in a Hostile Environment". In: *Annual Cryptology Conference*. Springer, 2014 (cit. on p. 7).

[26] J. Camenisch and A. Lehmann. "Privacy-Preserving User-Auditable Pseudonym Systems". In: *2017 IEEE European Symposium on Security and Privacy (EuroSP)*. 2017, pp. 269–284. DOI: 10.1109/EuroSP.2017.36 (cit. on p. 34).

[27] J. Camenisch, A. Lehmann, and G. Neven. "Optimal Distributed Password Verification". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015 (cit. on p. 7).

[28] J. Camenisch et al. "Oblivious PRF on Committed Vector Inputs and Application to Deduplication of Encrypted Data". In: *International Conference on Financial Cryptography and Data Security*. Springer, 2019 (cit. on p. 7).

[29] J. Camenisch et al. *Oblivious PRF on Committed Vector Inputs and Application to Deduplication of Encrypted Data*. Cryptology ePrint Archive, Paper 2019/438. 2019. URL: https://eprint.iacr.org/2019/438 (cit. on p. 34).

[30] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001 (cit. on pp. 13, 49).

[31] R. Canetti et al. *Reusable Fuzzy Extractors for Low-Entropy Distributions*. Cryptology ePrint Archive, Paper 2014/243. 2014. URL: https://eprint.iacr.org/2014/243 (cit. on p. 19).

[32] S. Casacuberta, J. Hesse, and A. Lehmann. "SoK: Oblivious Pseudorandom Functions". In: *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&ampP)*. IEEE, 2022-06. DOI: 10.1109/eurosp53844.2022.00045. URL: https://doi.org/10.1109/eurosp53844.2022.00045 (cit. on pp. 1, 6, 24, 30, 31, 33, 77).

[33] M. Chase and P. Miao. "Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF". In: *Advances in Cryptology – CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. Vol. 12172. Lecture Notes in Computer Science (LNCS). Springer, Cham, 2020, pp. 34–63. DOI: 10.1007/978-3-030-56877-1_2 (cit. on p. 7).

[34] E. Chattopadhyay. "Guest Column: A Recipe for Constructing Two-Source Extractors". In: *SIGACT News* 51.2 (2020), pp. 38–57. DOI: 10.1145/3406678.3406688. URL: https://doi.org/10.1145/3406678.3406688 (cit. on pp. 20, 71).

[35] S. Chaudhry et al. "A secure and efficient authenticated encryption for electronic payment systems using elliptic curve cryptography". In: *Electronic Commerce Research* 16 (2016), pp. 113–139. DOI: 10.1007/s10660-015-9192-5. URL: https://doi.org/10.1007/s10660-015-9192-5 (cit. on p. 2).

[36] D. Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology*. Springer, 1983, pp. 199–203 (cit. on p. 51).

[37] B. Chor and O. Goldreich. "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity". In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. Vol. 17. 1985-11, pp. 429–442. ISBN: 0-8186-0644-4. DOI: 10.1109/SFCS.1985.62 (cit. on pp. 20, 72).

[38] Cloudflare. *How Lava Lamps Help Keep the Internet Secure*. Accessed: June 24, 2024. 2020. URL: https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/ (cit. on p. 18).

[39] *Cryptographic security protocols: TLS*. URL: https://www.ibm.com/docs/en/ibm-mq/9.4?topic=overview-cryptographic-security-protocols-tls (visited on 2024-07-19) (cit. on p. 26).

[40] J. Daemen and V. Rijmen. *The Design of Rijndael: The Advanced Encryption Standard (AES)*. 2nd. Information Security and Cryptography. Springer, 2020. ISBN: 978-3-662-60769-5. DOI: 10.1007/978-3-662-60769-5. URL: https://link.springer.com/book/10.1007/978-3-662-60769-5 (cit. on pp. 17, 21).

[41] P. Das, J. Hesse, and A. Lehmann. "DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password". In: *Cryptology ePrint Archive* (2020) (cit. on pp. 7, 34, 49).

[42] A. Davidson, N. Sullivan, and C. Wood. *Oblivious pseudorandom functions (OPRFs) using prime-order groups*. Tech. Rep. Internet Engineering Task Force, 2019. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-voprf/ (cit. on pp. 1, 6, 30, 32).

[43] A. Davidson et al. "Privacy Pass: Bypassing Internet Challenges Anonymously". In: *Proceedings on Privacy Enhancing Technologies* (2018) (cit. on pp. 6, 7, 34, 49).

[44] A. Davidson. *Privacy Pass - "The Math"*. Cloudflare Blog. 2017. URL: https://blog.cloudflare.com/privacy-pass-the-math/ (visited on 2023-05-24) (cit. on p. 6).

[45] G. T. Davies et al. "Security Analysis of the WhatsApp End-to-End Encrypted Backup Protocol". In: *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference*. Vol. Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science (LNCS). Springer, 2023, pp. 330–361 (cit. on pp. 2, 7).

[46] E. De Cristofaro, P. Gasti, and G. Tsudik. "Fast and Private Computation of Cardinality of Set Intersection and Union". In: *Cryptology and Network Security*. Springer Berlin Heidelberg, 2012, pp. 218–231 (cit. on p. 53).

[47] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. 2008-08. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/rfc/rfc5246 (cit. on pp. 23, 26).

[48] W. Diffie and M. E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on p. 26).

[49] Y. Dodis et al. "On the (Im)Possibility of Cryptography with Imperfect Randomness". In: *45th Annual IEEE Symposium on Foundations of Computer Science*. 2004, pp. 196–205. DOI: 10.1109/FOCS.2004.44 (cit. on p. 19).

[50] Y. Dodis and J. Spencer. "On the (Non)Universality of the One-Time Pad". In: *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2002-11, pp. 376–385 (cit. on p. 19).

[51] Y. Dodis and Y. Yao. *Privacy with Imperfect Randomness*. Cryptology ePrint Archive, Paper 2014/623. 2014. URL: https://eprint.iacr.org/2014/623 (cit. on p. 19).

[52] Y. Dodis et al. *Differential Privacy with Imperfect Randomness*. Cryptology ePrint Archive, Paper 2012/435. 2012. URL: https://eprint.iacr.org/2012/435 (cit. on p. 19).

[53] B. Dowling et al. "A Cryptographic Analysis of the TLS 1.3 Handshake Protocol". In: *Journal of Cryptology* 34.4 (2021-10), p. 37 (cit. on pp. 18, 25).

[54] W. Easttom. "Virtual Private Networks, Authentication, and Wireless Security". In: *Modern Cryptography: Applied Mathematics for Encryption and Information Security*. Cham: Springer International Publishing, 2021, pp. 299–317. ISBN: 978-3-030-63115-4. DOI: 10.1007/978-3-030-63115-4_14. URL: https://doi.org/10.1007/978-3-030-63115-4_14 (cit. on p. 2).

[55] S. Faust, C. Hazay, and D. Venturi. "Outsourced Pattern Matching". In: *International Colloquium on Automata, Languages, and Programming*. Springer, 2013, pp. 545–556 (cit. on p. 7).

[56] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010 (cit. on p. 26).

[57] M. J. Freedman et al. "Keyword search and oblivious pseudorandom functions". In: *Theory of Cryptography Conference*. Springer, 2005, pp. 303–324 (cit. on pp. 1, 7, 30, 34).

[58] A. O. Freier, P. Karlton, and P. C. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101. 2011-08. DOI: 10.17487/RFC6101. URL: https://www.rfc-editor.org/rfc/rfc6101 (cit. on p. 26).

[59] O. Goldreich. *Foundations of Cryptography: Volume 1 - Basic Tools*. Cambridge University Press, 2001. ISBN: 9780521791725 (cit. on p. 23).

[60] O. Goldreich, S. Goldwasser, and S. Micali. "How to Construct Random Functions". In: *Journal of the ACM (JACM)* 33.4 (1986), pp. 792–807 (cit. on pp. 23, 26).

[61] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*. Providence, Rhode Island, USA: ACM, 1985, pp. 291–304. DOI: 10.1145/22145.22178 (cit. on p. 33).

[62] M. Green. *Let's Talk about PAKE*. Cryptography Engineering Blog. 2018. URL: https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/ (visited on 2023-06-15) (cit. on pp. 7, 35, 36).

[63] C. Hazay. "Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs". In: *J. Cryptology* 31 (2018), pp. 537–586. DOI: 10.1007/s00145-017-9263-y (cit. on p. 34).

[64] C. Hazay and Y. Lindell. "Efficient Protocols for Set Intersection and Pattern Matching with Security against Malicious and Covert Adversaries". In: *Theory of Cryptography Conference*. Springer, 2008, pp. 155–175 (cit. on pp. 6, 7, 30, 34).

[65] A. Hulsing et al. "Post-Quantum WireGuard". In: *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2021-05, pp. 304–321 (cit. on p. 25).

[66] S. Jarecki et al. "TOPPSS: Cost Minimal Password-Protected Secret Sharing Based on Threshold OPRF". In: *International Conference on Applied Cryptography and Network Security*. Springer, 2017 (cit. on pp. 7, 34, 49).

[67] S. Jarecki, A. Kiayias, and H. Krawczyk. "Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 233–253 (cit. on pp. 30, 34, 49, 50, 52).

[68] S. Jarecki, H. Krawczyk, and J. Resch. "Updatable Oblivious Key Management for Storage Systems". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 379–393 (cit. on p. 30).

[69] S. Jarecki, H. Krawczyk, and J. Xu. *OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks*. Cryptology ePrint Archive, Paper 2018/163. https://eprint.iacr.org/2018/163. 2018. URL: https://eprint.iacr.org/2018/163 (cit. on pp. 6, 7, 34, 36, 49).

[70] S. Jarecki, H. Krawczyk, and J. Xu. "On the (In)Security of the Diffie-Hellman Oblivious PRF with Multiplicative Blinding". In: *Public-Key Cryptography – PKC 2021: 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part II*. Springer-Verlag, 2021, pp. 380–409 (cit. on pp. 1, 34, 50).

[71] S. Jarecki and X. Liu. "Fast Secure Computation of Set Intersection". In: *International Conference on Security and Cryptography for Networks*. Springer, 2010, pp. 418–435 (cit. on pp. 30, 54).

[72] S. Jarecki and X. Liu. "Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection". In: *Theory of Cryptography Conference*. Springer, 2009, pp. 577–594 (cit. on pp. 7, 30, 34).

[73] D. Kales et al. "Mobile Private Contact Discovery at Scale". In: *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019, pp. 1447–1464 (cit. on pp. 7, 34).

[74] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. 2nd ed. CRC Press, 2015 (cit. on pp. 2, 3, 12, 15, 16, 22–24, 28, 44).

[75] Á. Kiss et al. "Private Set Intersection for Unequal Set Sizes with Mobile Applications". In: *Proceedings of PETS'17* (2017), pp. 206–226 (cit. on pp. 6, 7, 23, 34).

[76] N. Koblitz. "Elliptic Curve Cryptosystems". In: *Mathematics of Computation* 48.177 (1987), pp. 203–209. DOI: 10.1090/S0025-5718-1987-0866109-5 (cit. on pp. 17, 22).

[77] V. Kolesnikov et al. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 818–829 (cit. on pp. 6, 7, 30, 34).

[78] V. Kolesnikov et al. *Practical Multi-party Private Set Intersection from Symmetric-Key Techniques*. Cryptology ePrint Archive, Paper 2017/799. 2017. URL: https://eprint.iacr.org/2017/799 (cit. on p. 34).

[79] H. Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by T. Rabin. Vol. 6223. Lecture Notes in Computer Science. Heidelberg: Springer, 2010, pp. 631–648 (cit. on pp. 19, 23).

[80] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Updated by RFC 6151. RFC Editor, 1997-02. URL: https://www.rfc-editor.org/rfc/rfc2104 (cit. on pp. 22, 25).

[81] K. Lewi et al. *Oblivious Revocable Functions and Encrypted Indexing*. Tech. rep. 2022/1044. Cryptology ePrint Archive, 2022. URL: https://eprint.iacr.org/2022/1044 (cit. on pp. 2, 7).

[82] X. Li. "Improved Two-Source Extractors, and Affine Extractors for Polylogarithmic Entropy". In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 168–177. DOI: 10.1109/FOCS.2016.26 (cit. on p. 72).

[83] X. Li. *Non-Malleable Extractors, Two-Source Extractors and Privacy Amplification*. Cryptology ePrint Archive, Paper 2012/188. 2012. URL: https://eprint.iacr.org/2012/188 (cit. on p. 71).

[84] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[85] J. L. McInnes and B. Pinkas. "On the Impossibility of Private Key Cryptography with Weakly Random Keys". In: *Advances in Cryptology - CRYPTO '90*. Ed. by A. J. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin Heidelberg, 1991, pp. 421–435. DOI: 10.1007/3-540-38424-3_31 (cit. on p. 19).

[86] P. Miao et al. "Two-Sided Malicious Security for Private Intersection-Sum with Cardinality". In: *Advances in Cryptology – CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. Cham: Springer International Publishing, 2020, pp. 3–33. ISBN: 978-3-030-56877-1 (cit. on p. 34).

[87] D. Nagy. "On Digital Cash-Like Payment Systems". In: *Communications in Computer and Information Science*. Springer, 2007 (cit. on p. 2).

[88] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. Bitcoin.org, 2008. URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 2).

[89] M. Naor, B. Pinkas, and O. Reingold. "Distributed Pseudo-random Functions and KDCs". In: *Advances in Cryptology — EUROCRYPT '99*. Springer, 1999, pp. 327–346 (cit. on p. 50).

[90] National Institute of Standards and Technology. *Cryptography*. URL: https://www.nist.gov/cryptography (cit. on p. 2).

[91] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication (FIPS PUBS) 198-1. Washington, D.C.: U.S. Department of Commerce, 2008 (cit. on p. 25).

[92] National Security Agency. *National Security Agency (NSA)*. URL: https://www.nsa.gov/ (cit. on p. 2).

[93] N. Nisan and A. Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167. URL: https://theory.stanford.edu/~liyang/teaching/projects/hardness-vs-randomness.pdf (cit. on p. 23).

[94] NIST. *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. Tech. rep. SP 800-56A Rev. 3. Accessed: 2024-09-01. National Institute of Standards and Technology, 2018. URL: https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final (cit. on p. 28).

[95] T. Peacock et al. "Chapter e90 - Verifiable Voting Systems". In: *Computer and Information Security Handbook (Third Edition)*. Third Edition. Elsevier, 2013, e293–e315. ISBN: 978-0-12-803843-7. URL: https://www.sciencedirect.com/science/article/pii/B9780128038437000909 (cit. on p. 2).

[96] B. Pinkas et al. *Secure Two-Party Computation is Practical*. Cryptology ePrint Archive, Paper 2009/314. 2009. URL: https://eprint.iacr.org/2009/314 (cit. on p. 34).

[97] B. Pinkas et al. *Efficient Circuit-based PSI with Linear Communication*. Cryptology ePrint Archive, Paper 2019/241. 2019. URL: https://eprint.iacr.org/2019/241 (cit. on p. 34).

[98] B. Pinkas et al. *SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension*. Cryptology ePrint Archive, Paper 2019/634. 2019. URL: https://eprint.iacr.org/2019/634 (cit. on p. 34).

[99] J. Pullman, K. Thomas, and E. Bursztein. *Protect Your Accounts From Data Breaches With Password Checkup*. 2019. URL: https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html (cit. on p. 7).

[100] R. Raz. "Extractors with weak random seeds". In: *Proceedings of the Annual ACM Symposium on Theory of Computing* (2005), pp. 11–20. ISSN: 0737-8017. DOI: 10.1145/1060590.1060593 (cit. on p. 72).

[101] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. 2018-08. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446 (cit. on p. 26).

[102] R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 4, 17, 21).

[103] R. L. Rivest and J. Schuldt. "Spritz—a spongy RC4-like stream cipher and hash function". In: *Proceedings of the 2014 International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2014, pp. 1–19. URL: https://people.csail.mit.edu/rivest/pubs/RS14.pdf (cit. on p. 21).

[104] P. Schwabe, D. Stebila, and T. Wiggers. "Post-Quantum TLS without Handshake Signatures". In: *ACM CCS 2020*. Ed. by J. Ligatti et al. ACM Press, 2020-11, pp. 1461–1480 (cit. on p. 25).

[105] C. E. Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656 (cit. on p. 20).

[106] N. I. of Standards and T. (NIST). *Recommendation for the Entropy Sources Used for Random Bit Generation*. Tech. rep. NIST SP 800-90B. National Institute of Standards and Technology (NIST), 2018-01. URL: https://doi.org/10.6028/NIST.SP.800-90B (cit. on p. 21).

[107] N. I. of Standards and T. (NIST). *Secure Hash Standard (SHS)*. Tech. rep. FIPS 180-4. National Institute of Standards and Technology, 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (cit. on p. 22).

[108] D. Stebila, S. Fluhrer, and S. Gueron. *Hybrid Key Exchange in TLS 1.3*. https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-05. draft-ietf-tls-hybrid-design-05. 2022-08 (cit. on p. 25).

[109] N. Tyagi et al. *A Fast and Simple Partially Oblivious PRF, with Applications*. Cryptology ePrint Archive, Paper 2021/864. 2021. URL: https://eprint.iacr.org/2021/864 (cit. on p. 34).

[110] S. P. Vadhan. *Pseudorandomness*. Vol. 7. Foundations and Trends in Theoretical Computer Science 1-3. Now Publishers Inc., 2012, pp. 1–336 (cit. on p. 20).

[111] S. Yilek et al. "When private keys are public: results from the 2008 Debian OpenSSL vulnerability". In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. IMC '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 15–27. ISBN: 9781605587714. DOI: 10.1145/1644893.1644896. URL: https://doi.org/10.1145/1644893.1644896 (cit. on p. 8).

[112] D. Zuckerman. "General weak random sources". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science* (1990), 534–543 vol.2 (cit. on p. 20).