Oblivious Pseudo-Random Functions via Garbled Circuits

Master's Thesis of

Sebastian Faller

1939715

at the Department of Informatics
Institute for Theoretical Computer Science (ITI)

Reviewer:	Prof. Dr. Jörn Müller-Quade
Second reviewer:	Prof. Dr. Thorsten Strufe
Advisor:	M.Sc. Astrid Ottenhues
Second advisor:	M.Sc. Johannes Ernst

3. September 2021 – 3. March 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 03.03.2022

. . . .

(Sebastian Faller)

Abstract

An Oblivious Pseudo-Random Function (OPRF) is a protocol that allows two parties – a server and a user – to jointly compute the output of a Pseudo-Random Function (PRF). The server holds the key for the PRF and the user holds an input on which the function shall be evaluated. The user learns the correct output while the inputs of both parties remain private. If the server can additionally prove to the user that several executions of the protocol were performed with the same key, we call the OPRF verifiable.

One way to construct an OPRF protocol is by using generic tools from multi-party computation, like Yao's seminal garbled circuits protocol. Garbled circuits allow two parties to evaluate any boolean circuit, while the input that each party provides to the circuit remains hidden from the respective other party. An approach to realizing OPRFs based on garbled circuits was e.g. mentioned by Pinkas et al. (ASIACRYPT '09). But OPRFs are used as a building block in various cryptographic protocols. This frequent usage in conjunction with other building blocks calls for a security analysis that takes composition, i.e., the usage in a bigger context into account.

In this work, we give the first construction of a garbled-circuit-based OPRF that is secure in the universal composability model by Canetti (FOCS '01). This means the security of our protocol holds even if the protocol is used in arbitrary execution environments, even under parallel composition. We achieve a passively secure protocol that relies on authenticated channels, the random oracle model, and the security of oblivious transfer. We use a technique from Albrecht et al. (PKC '21) to extend the protocol to a verifiable OPRF by employing a commitment scheme. The two parties compute a circuit that only outputs a PRF value if a commitment opens to the right server-key.

Further, we implemented our construction and compared the concrete efficiency with two other OPRFs. We found that our construction is over a hundred times faster than a recent lattice-based construction by Albrecht et al. (PKC '21), but not as efficient as the state-of-the-art protocol from Jarecki et al. (EUROCRYPT '18), based on the hardness of the discrete logarithm problem in certain groups. Our efficiency-benchmark results imply that – under certain circumstances – generic techniques as garbled circuits can achieve substantially better performance in practice than some protocols specifically designed for the problem.

Büscher et al. (ACNS '20) showed that garbled circuits are secure in the presence of adversaries using quantum computers. This fact combined with our results indicates that garbled-circuit-based OPRFs are a promising way towards efficient OPRFs that are secure against those quantum adversaries.

Zusammenfassung

Eine Oblivious Pseudo-Random Function (OPRF) ist ein Protokoll, dass es einem Server und einem Nutzer erlaubt, gemeinsam die Ausgabe einer Pseudozufallsfunktion (PRF) zu berechnen. Der Server besitzt den Schlüssel, unter welchem die Funktion ausgewertet wird. Der Nutzer besitzt einen Eingabewert, an dem die Funktion ausgewertet wird. Der Nutzer erhält die korrekte Ausgabe während keine Partei die Eingabe der anderen erfährt. Kann der Server dem Nutzer zusätzlich beweisen, dass in mehreren Protokollausführungen der selbe Schlüssel verwendet wurde, so nennen wir die OPRF verifizierbar. Eine Möglichkeit ein OPRF Protokoll zu konstruieren ist, generische Techniken aus dem Bereich der sicheren Mehrparteienberechnung, wie Yao's Garblet Circuits, zu verwenden. Garbled Circuits erlauben es zwei Parteien gemeinsam einen beliebigen boolschen Schaltkreis auszuwerten, wobei die Eingaben beider Parteien geheim bleiben. Die Möglichkeit, eine OPRF mithilfe von Garbled Circuits zu erhalten, wurde z.B. von Pinkas et al. (ASIACRYPT '09) erwähnt. Allerdings werden OPRFs oft als Baustein in größeren Protokollen verwendet. Dieser häufige Einsatz in Verbindung mit anderen Bausteinen erfordert eine Sicherheitsanalyse, die Komposition, also die Verwendung in größerem Kontext, mit einbezieht.

In dieser Arbeit geben wir die erste Konstruktion einer OPRF an, die auf Garbled Circuits basiert und deren Sicherheit gleichzeitig im Universal Composability-Modell von Canetti (FOCS '01) bewiesen ist. Das bedeutet, unsere Sicherheitsanalyse ist auch dann noch aussagekräftig, wenn das Protokoll in beliebigen Umgebungen, sogar unter paralleler Komposition eingesetzt wird. Wir erhalten ein passiv sicheres Protokoll, dass unter der Annahme von authentifizierten Kanälen, des Random Oracle Models und der Sicherheit eines Oblivious Transfer Protokolls, sicher ist. Wir setzen eine von Albrecht et al. (PKC '21) vorgeschlagene Technik ein, um unser Protokoll zu einer verifizierbaren OPRF zu erweitern. Wir verwenden dazu ein Commitment Verfahren. Die Parteien berechnen einen leicht veränderten Schaltkreis, der nur dann die PRF Ausgabe erzeugt, wenn sich ein Commitment auf den Schlüssel des Servers korrekt öffnen lässt.

Zusätzlichen haben wir unsere Konstruktion implementiert und vergleichen die Effizienz mit zwei weiteren OPRF Konstruktionen. Die Experimente zeigen, dass unsere OPRF mehr als 110-mal schneller ist, als die Gitter-basierte OPRF von Albrecht et al. (PKC '21). Unsere Konstruktion ist allerdings nicht so effizient wie die OPRF von Jarecki et al. (EUROCRYPT '18), die auf der Schwierigkeit der Berechnung diskreter Logrithmen basiert. Unsere Experimente zeigen, dass – unter bestimmten Umständen – generische Techniken wie Garbled Circuits eine wesentlich bessere Effizienz erreichen können, als speziell auf den Anwendungsfall zugeschnittene Protokolle. Büscher et al. (ACNS '20) haben gezeigt, dass Garbled Circuits sicher gegen Angreifer sind, die im Besitz von Quantencomputern sind. Nimmt man diese Tatsache mit unseren Ergebnissen zusammen, zeigt sich, dass Garbled Circuit-basierte OPRFs ein wichtiger Schritt auf dem Weg zu effizienten und gleichzeitig gegen derartige Quantenangreifer sicheren OPRFs sind.

Contents

List of Figures

1. Introduction

A Pseudo-Random Function (PRF) is a function $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^l$, where F takes a key $k \in \{0,1\}^m$ and an input value $x \in \{0,1\}^n$ and outputs a value $y \in \{0,1\}^l$, and where $m, n, l \in \mathbb{N}$ are parameters that depend on the security parameter $\lambda \in \mathbb{N}$. If the key is chosen uniformly at random, the output of the function must be indistinguishable from a uniformly random value. However, such a conventional PRF must be evaluated by a single party, which knows k as well as x. In certain settings, a stronger primitive might be desirable. Imagine two parties where one party holds k and the other party holds x. If the two parties want to compute a pseudo-random value but hide their inputs from each other, a normal PRF is no solution. One party would need to send its input to the other party in order to evaluate the PRF. This problem can be tackled by using an Oblivious Pseudo-Random Function (OPRF). An OPRF for a certain PRF consists of two parties that interact to jointly compute an output of the PRF. One party called the server holds the key k of the PRF and the other party called the user holds the input value x. In the end, the user learns the output value $y = \mathsf{F}_k(x)$, but nothing about the key k. The server obtains no additional information from the interaction. In particular, it learns nothing about the user's input x. The just described notion of OPRF notion is also called *strong* OPRF. For certain applications, it might be too restrictive. Instead of demanding that the user learns only the output value $y = \mathsf{F}_k(x)$ but nothing about the key k, one can demand the following: The user learns nothing about the key k that would help in calculating further PRF outputs $y' = \mathsf{F}_k(x')$ for $x \neq x'$. This is called a *weak* or *relaxed* OPRF. An OPRF is called verifiable, if the server can prove to the user, that the "right" key k was used. More precisely, the server convinces the user that the same server key was used in all interactions with the client. Figure 1.1 depicts the general idea behind an OPRF execution.

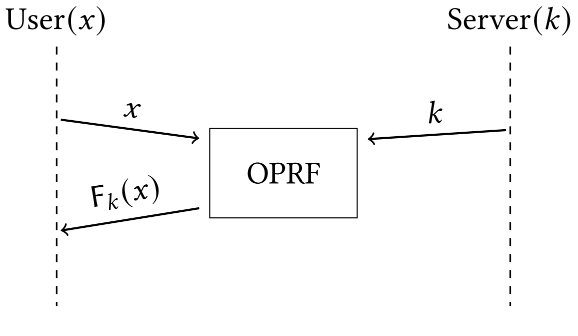


Figure 1.1.: Sketch of the Oblivious Pseudo-Random Function (OPRF) Functionality.

Motivation Conventional PRFs are a very useful and well-established building block. They can be used e.g. for digital signatures and message authentication [BG90], checking the correctness of memory [Blu+91], tracing sources of data leakage [CFN94], and many more. However, there are scenarios in which the additional possibility to evaluate the PRF obliviously between two parties is beneficial. Consider for example a typical password authentication on a website. Nowadays, most websites avoid storing the password of a client in the clear. Instead, a random value, the so-called "salt", is generated. The user's password and the salt are hashed and only the hash value and the salt are stored by the webserver. To authenticate itself, the user sends the password to the server, who in turn recomputes the hash and compares it to the stored value. This is depicted in Figure 1.2.

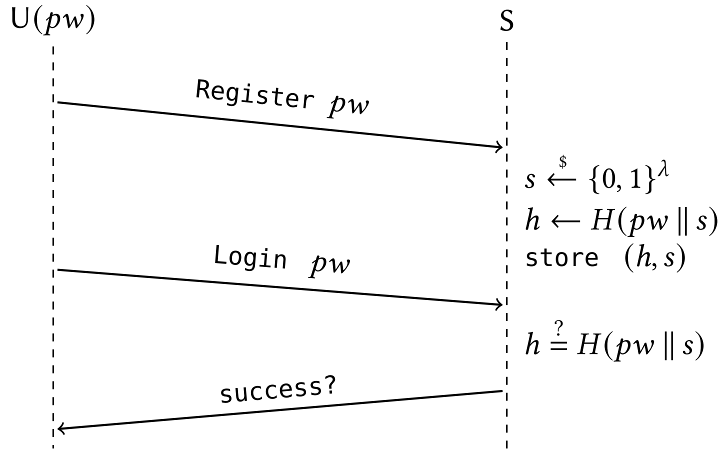

U(pw)
S
Register pw
$s \xleftarrow{\$} \{0,1\}^\lambda$
$h \leftarrow H(pw \| s)$
store (h, s)
Login pw
$h \overset{?}{=} H(pw \| s)$
success?

Figure 1.2.: Usual Authentication With Password and Salt.

Clearly, the user has to send its password over the network every time the user wants to authenticate itself to the server. If an adversary can eavesdrop on this authentication, the cleartext password allows the adversary to try the password at another service, where the user likely has used a similar password. Even if the communication is secured via a Transport Layer Security (TLS) channel, the password might get stolen, e.g., as TLS-certificates of servers can get stolen. This problem can be avoided by using OPRFs.

The idea to use an OPRF for password-authentication lies at the heart of a construction, called *OPAQUE* [JKX18]. OPAQUE is an asymmetric Password Authenticated Key Exchange (aPAKE) that allows a user to authenticate itself to a server using a password. If the authentication is successful, an ephemeral session key is exchanged. The whole interaction only requires the user to send its password in the clear once, at the first registration. Roughly speaking, the gist of OPAQUE is to let the user receive a pseudo-random value $y = \mathsf{F}_k(x)$, where the key k comes from the server and the input x to the PRF is the user's password. This pseudo-random value is then employed by the user to decrypt further information from the server, i.e., asymmetric keys for an authenticated key exchange that were generated at registration.

Additionally, there is a plethora of other interesting applications for OPRFs, including private set intersection [JL09], password-protected secret sharing [JKK14; Jar+16], secure keyword search [Fre+05], secure data de-duplication [KBR13], and privacy-preserving lightweight authentication mechanisms [Dav+18].

As OPRFs are often used as building blocks to solve more complex cryptographic tasks, it would be desirable to have a security analysis that takes composition into account. The

Universal Composability (UC) model by Canetti [Can01] offers such security guarantees. That means in particular that security proofs in the UC-model remain meaningful, even if the analyzed protocol is used in arbitrary contexts and might be executed in parallel or with correlated inputs. Over the last years, several works investigated the security of OPRF protocols in that model. To the best of our knowledge, Jarecki, Kiayias, and Krawczyk [JKK14] were the first to define an ideal verifiable OPRF functionality in the UC-model. Subsequent works [Jar+16; JKX18; BKW20] enhanced and modified the definition. Jarecki et al. [Jar+16] and Jarecki, Krawczyk, and Xu [JKX18] dispensed with the verifiability property.

Realization via Garbled Circuits The above described strong OPRF functionality can actually be seen as a problem in the field of Multi-Party Computation (MPC). The goal of an OPRF is to securely compute the two-party functionality

$$(k, x) \mapsto (\perp, \mathsf{F}_k(x)),$$

where $\perp$ denotes that the server receives no output. One of the most famous protocols to solve this task is Yao's garbled circuits [Yao86]. Garbled circuits are a vivid field of research. The main idea is that two parties, Alice and Bob, want to compute a commonly known boolean circuit C on two input strings $x, y \in \{0,1\}^*$, where x is only known to Alice and y is only known to Bob. At the end of the protocol, both parties should learn the output of $C(x, y)$, i.e., the circuit evaluated on the two input strings. However, Alice should "not learn anything" about y and similarly, Bob should not learn anything about x.

To the best of our knowledge, the apparent idea to realize an OPRF by using garbled circuits was first described by Pinkas et al. [Pin+09]. However, we believe that a second look at the idea is beneficial for several reasons:

- OPRFs are usually used as a building block to build more powerful cryptographic protocols, see [JL09; JKK14; Jar+16; Fre+05; KBR13; Dav+18]. While e.g. [LP07] consider security of garbled circuits in the simulation-based model of [Can98], we prefer a treatment in the more current Universal Composability (UC) framework of [Can01]. UC security offers strong security guarantees under composition. We can also rely on more recent work on the formulation of an idealized OPRF functionality by [Jar+16; JKX18]. We even argue – without formal proof – why the construction of [Pin+09] does not satisfy the OPRF notion of [JKX18], i.e., does not UC-realize their ideal OPRF functionality.

- The recent advantages in the field of MPC brought further improvements on the concrete efficiency of garbled circuits, most notably, the work of [ZRE15]. This allows for even more efficient implementations of the mentioned OPRF protocol than described by [Pin+09].

- Pinkas et al. [Pin+09] do not consider verifiability of the OPRF. We adapt ideas from [Alb+21] to achieve a verifiable OPRF.

Another point that makes an OPRF construction from garbled circuits interesting is the fast progress in the field of quantum computing over the last years. Recently, Arute et al. [Aru+19] claimed that they reached quantum supremacy for the first time. This means they computed a problem on a quantum computer that would have taken a significantly larger amount of time on a classical computer. Some researches suggest that practical quantum computing could be possible in the next two decades [Mos18; Bau+16]. Even if these estimates were over-optimistic, they make further progress in this field of research conceivable. Quantum computers pose serious threats to classical cryptographic constructions because the seminal work of Shor [Sho94] shows that the discrete logarithm problem and the integer factorization problem can be solved efficiently by a quantum computer. Therefore, it is necessary to further investigate post-quantum secure cryptographic building blocks, i.e., building blocks that are secure against adversaries using quantum computers.

Büscher et al. [Büs+20] showed that garbled circuits are secure in the presence of adversaries, using quantum computers – so-called quantum adversaries. Intuitively, this is because garbled circuits rely on symmetric cryptography and Oblivious Transfer (OT) and quantum adversaries have no substantial advantage over conventional computers in breaking those primitives. Thus, garbled circuits are promising for providing a way of achieving post-quantum secure OPRFs. Over the last decades, several works improved the efficiency of garbled circuits dramatically, see Section 2.6. It is therefore an interesting research question, whether a garbled-circuit-based OPRF will perform comparably or even better than constructions that are directly based on presumably post-quantum secure assumptions, as the lattice-based construction by Albrecht et al. [Alb+21].

1.1. Contribution

In this work, we construct the first garbled-circuit-based OPRF that is secure under universal composition [Can01]. We argue informally why the garbled-circuit-based OPRF by [Pin+09] does not UC-realize ideal OPRF functionalities like [Jar+16; JKX18] and show how to overcome their limitation by introducing a further programmable random-oracle as in [JKX18]. We implemented the protocol and compared its concrete efficiency to the OPRF protocols of [JKX18] and [Alb+21].

Technical Overview From a high point of view, our protocol follows the idea of Pinkas et al. [Pin+09] that can be sketched as follows: If the server and the user participate in a secure two-party computation, where the jointly computed circuit is a PRF, the resulting protocol is an OPRF. However, we additionally introduced two hash functions. The first hash function allows the user to hash an input string of arbitrary length to the input size of the PRF. The second hash function is applied to the output of the garbled circuit and to the original user input. Both hash functions will be modeled as random oracles. The random oracles are crucial for the security proof, as both allow the simulator in the proof to obtain information about the current simulated execution. But even more importantly, we will need to program the second random oracle in certain situations. Roughly speaking, this is because ideal OPRF functionalities in the style of [JKX18] compare the outputs of the OPRF protocol with truly random values. But the UC-framework requires that the compared output values are indistinguishable, even if the OPRF is used as a building block in bigger contexts. That “bigger context” is modeled in the UC-framework by the so-called "environment" machine. But if the environment somehow knew the input x and the key k, merely computing a PRF as $\mathsf{F}_k(x)$ is completely deterministic. Thus, the simulator in the proof must be able to "adjust" the output, so it still "looks like the random output" of the ideal functionality. That can be done by programming the second random oracle. We will elaborate further on this in Section 3.4.

An execution of our protocol can be sketched as follows:

- The server chooses a uniformly random key k.
- The user hashes its input p and receives $h = H_1(p)$. It then requests a garbled circuit from the server by sending Garble to the server.
- The server garbles the circuit of a PRF and creates input labels for its key as well as for each possible input bit of the user. The server sends the garbled circuit, the key labels, and additional information that is needed to evaluate the circuit to the user.
- The user and the server jointly execute a 1-out-of-2 OT for each input bit of the user. The user sends the respective bit as choice bit and the server sends the two possible labels as the message. The user obtains only the labels for his input h.
- The user evaluates the garbled circuit on the labels for his input h and the labels for the server's key k. It receives an output y and hashes $H_2(p, y) = \rho$. The user outputs ρ.

This is also depicted graphically in Figure 5.1. We prove that the protocol from above UC-realizes an ideal OPRF functionality. We use a slightly simplified version of the functionality from [JKX18] for our proof. This means for instance, that our protocol can be used directly to instantiate the Password-Protected Secret Sharing protocol from [Jar+16].

To achieve verifiability, we use a technique proposed by Albrecht et al. [Alb+21]. We assume that the server publishes a commitment c on his key as "identificator" of its key. Now, we do not only garble the circuit of a PRF but a circuit that outputs the PRF output only if the "right key is used". The circuit takes the user's input and the commitment c as inputs from the user. The server provides its key and the opening information for c as input to the circuit. The new circuit calculates the PRF output, but only if the provided commitment correctly opens to k. As this verification of the commitment is "hard-wired" into the garbled circuit, the user still learns no additional information about k. But it can be sure that the received output is from the server that can open c.

Concrete Efficiency For the implementation, we used a C++ framework, called the *EMP-Toolkit* from Wang, Malozemoff, and Katz [WMK16]. We answer the question of how well our OPRF performs in comparison to the current state-of-the-art protocol, called *2HashDH*, by [JKK14; Jar+16; JKX18] and the lattice-based protocol by Albrecht et al. [Alb+21]. We assess the efficiency of the implementation in terms of running time and communication cost, i.e., the amount of data that has to be sent over the network. We performed our experiments on a conventional consumer laptop and did not take network latency into account. Our experiments show a noticeable gap in running time to the lattice-based construction of [Alb+21]. Our construction is over 110 times faster than the lattice-based protocol. As we explain in Section 5.3, this comparison has to be taken with a grain of salt. The experiments further show that 2HashDH by [JKK14; Jar+16; JKX18] is still about 50 times faster than our construction and requires less than 100 B of communication. This is not surprising as the protocol merely needs to exchange two points of an elliptic curve. However, with a running time of about 65 ms and traffic of about 250 kB our protocol is still in a reasonable efficiency range.

Implications of the Results Our experiments show that even though we employed the "generic" garbled circuit protocol, the resulting construction was still significantly more efficient than a special-purpose protocol based on lattices. This is somewhat surprising as garbled circuits allow to evaluate *any* boolean circuit privately. The main reason for this might be that garbled circuits are a matured cryptographic tool that was optimized several times, see Section 2.6, while Albrecht et al. [Alb+21] claim that their protocol is the first lattice-based Verifiable Oblivious Pseudo-Random Function (VOPRF). However, to reach a reasonable range of efficiency, there still seems to be a long way to go for lattice-based OPRFs.

Contrarily, it is plausible that our garbled-circuit-based construction is secure in the presence of adversaries with quantum computers, i.e., post-quantum secure, if an appropriate post-quantum secure OT protocol is chosen. The post-quantum security of garbled circuits was formally proven by Büscher et al. [Büs+20], which makes the post-quantum security of our protocol conceivable, even though we left a formal proof to future work. Considering the benchmark results, we see garbled-circuit-based OPRFs as promising candidates for practically efficient OPRFs that are secure in the presence of adversaries with quantum computers.

1.2. Related Work

First, we give a quick overview of other OPRF constructions in the literature. We divided them into three categories, depending on the underlying techniques of the protocols.

1.2.1. Diffie-Hellman-Based OPRFs

It is a well-known fact that a PRF can be constructed from a Pseudo-Random Generator (PRG) by using a tree construction, see for instance [BS20]. However, as this construction is not necessarily efficient, it is rather of theoretical interest. More efficient PRF constructions rely on the computational hardness of certain problems. We will first focus on PRFs that assume the hardness of variations of the Diffie-Hellman assumption. To the best of our knowledge, there are three such PRF constructions for which there is an associated OPRF protocol in the literature.

We start with the PRF, introduced by [Jar+16; JKK14; JKX18]. It is the most important of the Diffie-Hellman-based PRFs for this work. The underlying PRF can be formulated as

$$f_k^{\text{2HashDH}}(x) = H_2(p, H_1(x)^k),$$

where $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^n$ are modeled as random oracles and $\mathbb{G}$ is a group of prime order q for which a "one-more" version of the Decisional

Diffie-Hellman Assumption (DDH) assumption holds. The corresponding OPRF protocol, presented in [Jar+16; JKK14; JKX18] uses a technique which is sometimes referred to as "blinded exponentiation". This technique was first used in the context of blind signature, see [Cha83]. The main idea is that the user chooses some random $r \in \mathbb{Z}_q$ and sends g^r to the server. In turn, the server calculates $b := (g^r)^k$ and sends it back to the user. As the user knows r, it can calculate $b^{1/r} = g^k$. Thus, it received g^k without revealing the actual value of g to the server. By combining this idea with the two random oracles, one gets the protocol 2HashDH, depicted in Figure 5.2. This protocol is extremely efficient and the security is analyzed by [Jar+16; JKK14; JKX18] in the UC-framework by [Can01]. There is an ongoing effort to standardize this protocol by the Crypto Forum Research Group. See [Dav+22] for the current draft.

Another PRF was introduced by Naor and Reingold [NR04, Construction 4.1]. It is defined as follows: Let p, q be primes such that $q \mid p - 1$. Let $n \in \mathbb{N}$, $k = (a_0, \dots, a_n) \in \mathbb{Z}_q^{n+1}$ and $g \in \mathbb{Z}_p$ be an element of order q. The *Naor-Reingold PRF* with key k on input $x = (x_1, \dots, x_n) \in \{0,1\}^n$ is defined as

$$f_k^{\mathrm{NR}}(x) = g^{a_0 \cdot \prod_{i=1}^{n} a_i^{x_i}}.$$

Freedman et al. [Fre+05] proposed a constant-round OPRF protocol for the Naor-Reingold PRF that uses OT and the idea of *blinded exponentiation*, similar to [Jar+16; JKX18].

The third PRF, introduced by Dodis and Yampolskiy [DY05, Sec. 4.2] is defined as follows: Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order q and $k \in \mathbb{Z}_q^*$ be uniformly random. The *Dodis-Yampolskiy* PRF on input $x \in \mathbb{Z}_q^*$ is defined as

$$f_k^{\mathrm{DY}}(x) = g^{1/(x+k)}.$$

The security of the PRF is based on the so-called Decisional q-Diffie-Hellman Inversion Problem (q-DHI). Jarecki and Liu [JL09] and Belenkiy et al. [Bel+08] gave protocols to obliviously evaluate the above PRF. Both protocols employ a homomorphic encryption scheme, e.g. Paillier [Pai99].

All three Diffie-Hellman-Based OPRFs share the limitation that Shor's algorithms [Sho94] will render them insecure if sufficiently strong quantum computers become available. Additionally, the security proofs of [JL09; Bel+08] and [Fre+05] do not consider composition. In particular, they do not analyze their protocols in the UC-model of Canetti [Can01], as [Jar+16; JKK14; JKX18] and we do in our work.

1.2.2. MPC-Based OPRFs

The second category of OPRF protocols relies on techniques from Multi-Party Computation (MPC).

Pinkas et al. [Pin+09] argue that it is possible to realize an OPRF by using Yao's garbled circuits, see Section 2.6.3. Garbled circuits allow two parties to jointly evaluate any boolean circuit, while the input of each party is hidden from the respective other party. If the calculated circuit is a description of a PRF, the resulting output is the desired pseudo-random value. The privacy requirement for the OPRF is satisfied as the garbled circuit protocol guarantees the privacy of the inputs. Pinkas et al. [Pin+09] do not give a formal proof of security. However, they refer to the general proof for garbled circuit security in the presence of active adversaries of Lindell and Pinkas [LP07]. The simulation-based proof of [LP07] uses the framework of Canetti [Can98] that even considers composition to a certain extent. Albrecht et al. [Alb+21] sketch an idea of how to achieve verifiability from a garbled-circuit-based OPRF.

Kolesnikov et al. [Kol+16] choose a different MPC-based approach. They use efficient OT extensions, see [Ish+03], to instantiate something close to an OPRF protocol. The security notion they define is called batched, related-key OPRF (BaRK-OPRF). This notion is very similar to usual OPRFs. However, there are certain differences. The word "batched" means that the user can query pseudo-random output for $m \in \mathbb{N}$ different input values $r_1, \dots, r_m$. Each pseudo-random answer will be calculated using a different PRF key. "Related key" means that each PRF key is comprised of two components (k^*, k_i) and for every batch of input values $r_1, \dots, r_m$, i.e., for one protocol execution the first component of the PRF key stays the same. Therefore, all pseudo-random outputs were calculated under related keys.

Kolesnikov et al. [Kol+16] observe that an OT of random messages can be interpreted as a very simple OPRF. Concretely, if an OT-sender sends uniformly random messages $m_0, m_1 \in \{0,1\}^\lambda$ via OT and the receiver chooses one of them via a choice bit $b \in \{0,1\}$, the performed protocol is an OPRF for the PRF

$$F : \{0,1\}^{2\lambda} \times \{0,1\} \to \{0,1\}^\lambda; F_{(m_0,m_1)}(b) = m_b.$$

They improve the OT extension protocols from Ishai et al. [Ish+03] and Kolesnikov and Kumaresan [KK13] and achieve an efficient OT extension protocol for 1-out-of-n OT for exponentially large $n \in \mathbb{N}$. By combining this with the above idea, one gets an OPRF with input domain $\{1, \dots, n\}$. They analyze the security of their protocol in the UC-model of [Can01], as we did for our construction. A further similarity between our protocols is that both rely only on the security of OT and symmetric cryptography. In contrast to BaRK-OPRF, our construction does not enforce keys to be related. Server's can use completely independent keys for different OPRF executions.

1.2.3. OPRFs from Post-Quantum Assumptions

To the best of our knowledge, there are two OPRF constructions that directly rely on presumably post-quantum secure assumptions in the literature. Presumably post-quantum secure means that it is currently not believed by the cryptographic community, that a quantum computer is significantly more efficient in breaking those assumptions than a conventional computer. This distinguishes the post-quantum secure assumptions as e.g. Short Integer Solution (SIS), Learning With Errors (LWE), or the problem to find isogenies between supersingular elliptic curves from integer factorization or Discrete Logarithm (DLOG).

In fact, both constructions claim that they are even *verifiable* OPRFs. The first construction was proposed by Albrecht et al. [Alb+21]. It relies on the hardness of the decision version of the Ring-LWE assumption and the one-dimensional version of the SIS assumption, which was introduced by [BV15]. In contrast to our construction, their construction only needs two rounds of communication. However, the concrete efficiency is clearly worse, as relatively large parameters must be chosen.

The second construction was proposed by Boneh, Kogan, and Woo [BKW20] and is based on the hardness of certain isogeny problems. However, the construction was recently "broken" by Basso et al. [Bas+21]. They found that certain assumptions on isogenies made by [BKW20] did actually not hold. Basso et al. [Bas+21] further argue that there is no straightforward way to fix the construction. In conclusion, there is only the lattice-based construction from [Alb+21] left that directly relies on post-quantum secure assumptions.

However, there might be another possibility to achieve post-quantum secure OPRF, i.e., to use one of the MPC-based constructions. The rough idea would be to instantiate the protocol with a post-quantum secure OT protocol, as the rest of the security relies on symmetric encryption. This method might suffice, as quantum computers appear to have no substantial advantage in breaking symmetrical cryptography, see e.g. [BNS19; Amy+16]. While the security of the protocol from [Alb+21] provably holds in the Quantum-accessible Random Oracle Model (QROM) [Bon+11], i.e., in the presence of adversaries that can send superposition queries to the random oracle, we leave it to future work to formally prove the post-quantum security of one of the MPC-based constructions.

1.3. Outline

First, in Chapter 2 we will recall necessary definitions and introduce important techniques that we will apply later. Then we present and discuss our construction in Chapter 3. Of particular interest in that section might be the proof of security in the UC-model, which can be found in Section 3.5. In Chapter 4, we discuss which changes must be applied to our construction and to the security proof to achieve a VOPRF. We discuss the implementation of our protocol and the comparison of efficiency to other OPRFs in Chapter 5. We summarize our results and propose directions for future work on the topic in Chapter 6. Finally, in Appendix A, we present some techniques that are not necessary to understand this work, but that might be of interest to some readers.

2. Preliminary

2.1. Notation

We write λ for the security parameter. We always assume that all algorithms take λ as implicit parameter. We call a probabilistic Turing machine Probabilistic Polynomial Time (PPT), if its running time is bounded by a polynomial in λ. By $x \overset{\$}{\leftarrow} S$ we denote that x is chosen uniformly at random from the set S. We write $y \leftarrow A$ if the randomized algorithm $A(x)$ outputs y on input x. We write $x \parallel y$ for the concatenation of the strings x and y. We use $O(\cdot)$, $o(\cdot)$, $\Theta(\cdot)$, $\Omega(\cdot)$, and $\omega(\cdot)$ for asymptotic notation. We say a function is negligible in λ, if it asymptotically falls faster than the inverse of any polynomial in λ.

Particularly when describing simulators, we use $\langle\cdot\rangle$ for records, made by the simulator. We will use $\exists\langle x\rangle$ (or $\nexists\langle y\rangle$) to express that the simulator goes through its records and checks if there is a matching record $\langle x\rangle$ (or there is no matching record $\langle y\rangle$). Whenever the behavior of an ideal functionality on the receipt of a certain message is not explicitly defined, we assume that the functionality ignores the message.

Ausführlicher

2.2. Pseudo-Random Functions

A Pseudo-Random Function is function that produces "random looking" output values. More precisely, the function is indexed by a key k, sometimes called "seed". If the key is chosen uniformly at random, the function maps input values to output values in such a way, that it is indistinguishable whether the output values come from the pseudo-random function or from a truly random function. In that sense, one could see a PRF as a Pseudo-Random Generator "with random access" to the generated pseudo-random values. However, it is possible to construct a PRF from a PRG [BS20]. The security is defined via a PPT distinguisher $\mathcal{D}$ that either gets oracle access to $\mathsf{F}(k, \cdot)$ for some randomly chosen key $k \in \{0,1\}^m$ or to a truly random function RF. The goal of $\mathcal{D}$ to tell those situations apart.

Definition 1 (Pseudo-Random Function) [KL15, Def. 3.25] Let $n := n(\lambda)$ and $m := m(\lambda)$ be polynomial in λ. Let $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ be a function family such that there is a polynomial-time algorithm that takes $k \in \{0,1\}^m$ and $x \in \{0,1\}^n$ and outputs $\mathsf{F}(k, x)$.

We say PRF is a *pseudo-random function* if the advantage defined as

$$\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}}(\mathcal{D}, \lambda) := |*|\Pr_{k \overset{\$}{\leftarrow} \{0,1\}^m}\left[\mathcal{D}^{\mathsf{F}(k,\,\cdot\,)}(1^\lambda) = 1\right] - \Pr_{\mathsf{RF} \overset{\$}{\leftarrow} \{f:\{0,1\}^n \to \{0,1\}^n\}}\left[\mathcal{D}^{\mathsf{RF}(\,\cdot\,)}(1^\lambda) = 1\right]$$

is negligible for every PPT distinguisher $\mathcal{D}$, where the first probability is taken over uniform choices of $k \in \{0,1\}^m$ and the randomness of $\mathcal{D}$ and the second probability is taken over uniform choices of $\mathsf{RF} \in \{f : \{0,1\}^n \to \{0,1\}^n\}$ and the randomness of $\mathcal{D}$. □

Definition 2 (Pseudo-Random Permutation) [KL15, Sec. 3.5.1] Let $n := n(\lambda)$ and $m := m(\lambda)$ be polynomial in λ. Let $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ be a function family such that there is a polynomial-time algorithm that takes $k \in \{0,1\}^m$ and $x \in \{0,1\}^n$ and outputs $\mathsf{F}(k,x)$.

Let Perm_n denote the set of all permutations of length n. We say F is a *pseudo-random permutation* if the advantage defined as

$$\mathrm{Adv}^{\mathsf{PRP}}_{\mathsf{F}}(\mathcal{D},\lambda) := |*| \Pr_{k \xleftarrow{\$} \{0,1\}^m}\left[\mathcal{D}^{\mathsf{F}(k,\,\cdot\,)}(1^\lambda) = 1\right] - \Pr_{\mathsf{RP} \xleftarrow{\$} \mathrm{Perm}_n}\left[\mathcal{D}^{\mathsf{RP}(\,\cdot\,)}(1^\lambda) = 1\right]$$

is negligible for every PPT distinguisher $\mathcal{D}$, where the first probability is taken over uniform choices of $k \in \{0,1\}^m$ and the randomness of $\mathcal{D}$ and the second probability is taken over uniform choices of $\mathsf{RP} \in \mathrm{Perm}_n$ and the randomness of $\mathcal{D}$. $\square$

2.3. Commitment Schemes

Intuitively, a commitment scheme allows to create a value c, called the *commitment* on a message that hides the message but allows to only open the commitment to the original message. The commitment is opened by using the so-called *opening information*, which should be kept secret until the commitment has to be opened.

For the sake of simplicity, we will assume in this work that the messages, the commitments and the opening information are bit strings.

Definition 3 (Commitment Scheme) [BS20, Sec. 8.12] A Commitment Scheme consists of two efficient algorithms $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Unveil})$. For $n, l, t \in \Theta(\lambda)$, the Commit algorithm takes a message $m \in \{0,1\}^n$ and outputs $(c,r) \in \{0,1\}^l \times \{0,1\}^t$. We call c the *commitment on m* and r the *opening information.* The Unveil algorithm takes a commitment $c \in \{0,1\}^l$, a message $m \in \{0,1\}^n$ and the opening information $r \in \{0,1\}^t$ and outputs either 0 or 1, where we interpret output 1 as "c correctly opens to message m".

We require a commitment scheme to have *correctness.* By that we mean

$$\forall m \in \{0,1\}^n : \forall (c,r) \xleftarrow{\$} \mathsf{Commit}(m) : \Pr[\mathsf{Unveil}(c,k,r) = 1] = 1.$$

The commitment should not reveal any information about the committed message. One could also say that the commitment should *hide* the message from anyone who is not in possession of the opening information. We formalize this in the following definition. We define the security over a security experiment where the adversary $\mathcal{A}$ plays against a challenger C.

Definition 4 (Hiding) [BS20, Sec. 8.12] Let $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Unveil})$ be a commitment scheme. COM is *computationally hiding* if for every PPT adversary $\mathcal{A}$ we have

$$\mathrm{Adv}^{\mathsf{Hiding}}_{\mathsf{COM}}(\mathcal{A},\lambda) := \Pr\left[\mathrm{Exp}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda) = 1\right] \leq \mathrm{negl}(\lambda),$$

where $\mathrm{Exp}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is the experiment depicted in Figure 2.1 and $\mathrm{negl}(\cdot)$ is some negligible function and the probability is taken over the randomness of C and $\mathcal{A}$. $\square$

$$\mathrm{Exp}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$$

- $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^n$ to C.
- The challenger chooses a bit $b \in \{0,1\}$ uniformly at random. The challenger computer $(c, r) \leftarrow \mathsf{Commit}(m_b)$ and sends c to $\mathcal{A}$.
- $\mathcal{A}$ takes the input c and outputs a guess $b' \in \{0,1\}$.
- The experiment outputs 1 iff $b = b'$.

Figure 2.1.: The Hiding Experiment.

$$\mathrm{Exp}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$$

- $\mathcal{A}$ outputs (c, m_0, r_0, m_1, r_1).
- The experiment outputs 1 iff it holds that
  - $\mathsf{Unveil}(c, m_0, r_0) = 1$,
  - $\mathsf{Unveil}(c, m_1, r_1) = 1$,
  - $m_0 \neq m_1$.

Figure 2.2.: The Binding Experiment.

Additionally to the hiding property, we require that no efficient adversary should be able to lie about the message on which he committed. In other words, it should be hard for an adversary to first commit on some message and later open the commitment to another message. We will formalize this notion in the next definition.

Definition 5 (Binding) [BS20, Sec. 8.12] Let $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Unveil})$ be a commitment scheme. A COM is *computationally binding* if for every PPT adversary $\mathcal{A}$ that outputs a 5-tuple (c, m_0, r_0, m_1, r_1) with $c \in \{0,1\}^l$, $m_0, m_1 \in \{0,1\}^n$, $r_0, r_1 \in \{0,1\}^t$, we have

$$\mathrm{Adv}^{\mathsf{Binding}}_{\mathsf{COM}}(\mathcal{A}, \lambda) := \Pr\left[\mathrm{Exp}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda) = 1\right] \leq \mathrm{negl}(\lambda),$$

where $\mathrm{Exp}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is the experiment depicted in Figure 2.2 and $\mathrm{negl}(\cdot)$ is some negligible function and the probability is taken over the randomness of $\mathcal{A}$. □

There are also *statistical* and *perfect* variants of the notions of hiding and binding but we will not need them in this work.

2.4. Universal Composability

Often, cryptographic protocols are not used in isolation but are combined to serve a greater functionality. However, the security of protocols is not always conserved under composition. A classical result is for example that the parallel composition of two zero-knowledge protocols is in general not a zero-knowledge protocol [GK90]. *Universal Composability*, introduced by Canetti [Can01] is a notion of security that solves this problem. The original paper by Canetti [Can01] was revisited several times. In the following, we always refer to the version from 2020 [Can00]. Protocols that are secure in the UC model can be composed and preserve their security. This is stated more formally in the *composition theorem* from Canetti [Can00, Theo. 22].

The rough idea of the UC-security experiment is to compare an ideal world with the real world, similar to a "stand-alone" simulation-based proofs. This is conceptually visualized in Figure 2.3.

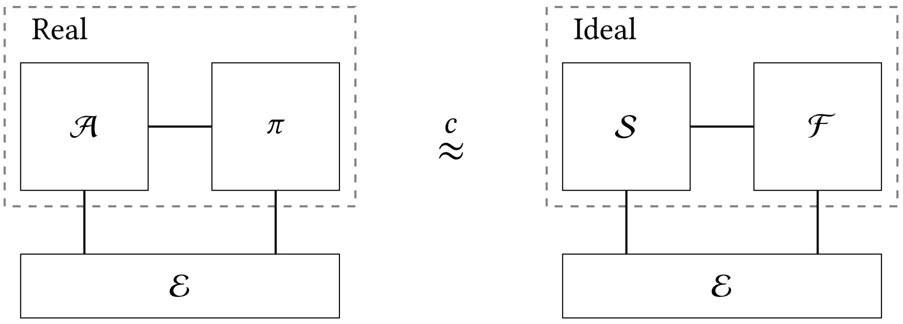


Figure 2.3.: Sketch of the Universal Composability Security Experiment.

In the ideal world, we do not regard the actual protocol but rather an idealized functionality $\mathcal{F}$. The gist is however, that all interactions between parties are orchestrated by a so-called environment machine $\mathcal{E}$. The environment machine can be thought of as the "bigger context" of the protocol execution, e.g., when the protocol is used as subroutine in another protocol. In contrast to a normal distinguisher in a stand-alone security notion, the environment can adaptively interact with the protocol parties.

In the real world, the environment machine $\mathcal{E}$ interacts with the real-world adversary $\mathcal{A}$ and with the real protocol parties of a protocol π.

In the ideal world, the protocol parties are replaced by "Dummy-Parties". These parties do nothing except forwarding all input directly to $\mathcal{F}$.

Additionally, the idealized functionality $\mathcal{F}$ and the environment $\mathcal{E}$ interact with a simulator $\mathcal{S}$, who plays the role of the real-world adversary. The job of $\mathcal{S}$ is to simulate an execution of π for $\mathcal{E}$ that looks like the real-world execution. If no PPT environment machine can tell both worlds apart, the protocol π UC-emulates (or UC-realizes) the ideal functionality $\mathcal{F}$. We will define this more formally (but still simplified) in the following. First, we define the notion of *UC-emulation*. This notion will in turn allow us to define the realization of an ideal functionality.

Definition 6 (UC-Emulation) [Can00, Def. 1] Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}(z)$ denote the random variable over the local random choices of all parties of π, of $\mathcal{E}$ and of $\mathcal{A}$ that describes an output of $\mathcal{E}$ on input $z \in \{0,1\}^*$ when running protocol π with adversary $\mathcal{A}$. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ denote the probability ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}(z)\}_{z\in\{0,1\}^*}$.

We say a protocol π *UC-emulates* an protocol ϕ, if for all PPT adversaries $\mathcal{A}$ there is an PPT simulator $\mathcal{S}$, such that for all environment machines $\mathcal{E}$ it holds that

$$\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}} \overset{c}{\approx} \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{E}}.$$

If this is the case, we write $\pi \geq \phi$. □

This notion captures more general situations than just the real-ideal comparison mentioned above. One can e.g. give two protocols π, ϕ with $\pi \geq \phi$, where π and ϕ are both "real-world" protocols. We now define what we mean exactly by realizing an ideal functionality.

Definition 7 (UC-Realization of a Functionality) [Can00, Def. 2] Let $\mathrm{IDEAL}_{\mathcal{F}}$ denote the protocol that consist of a machine $\mathcal{F}$, the ideal functionality, and m Dummy-Parties $\mathrm{D}_1, \ldots, \mathrm{D}_m$, where m is the number of parties that interact with $\mathcal{F}$. The Dummy-Parties only relay input to $\mathcal{F}$ and relay output from $\mathcal{F}$ to $\mathcal{E}$ and ignore all backdoor-tape messages. We say a protocol π *UC-realizes* a functionality $\mathcal{F}$, if π UC-emulates $\mathrm{IDEAL}_{\mathcal{F}}$.

We like to emphasize here that security in the UC-model is always defined relatively to an ideal functionality. Consequently, care must be taken, when $\mathcal{F}$ is specified.

Security in the UC model is a strong notion. However, even some very simple functionalities can not be achieved in the UC model without additional assumptions. The most famous result is the impossibility of bit-commitments in the plain model. Without additional assumptions, like a Common Reference String (CRS) or a Public-Key Infrastructure (PKI), there exists no protocol that UC-emulates the bit-commitment functionality $\mathcal{F}_{\mathrm{com}}$ [CF01, Theorem 6].

As mentioned in Section 1.2.1, [Jar+16; JKX18] defined ideal functionalities, describing the desired security of OPRFs in the UC model. They also constructed protocols that UC-realize the ideal functionalities. We will discus them in Section 2.7.

The Dummy Adversary Canetti [Can00] also shows, that Definition 7 can be simplified by using the *Dummy Adversary*. Instead of considering all PPT adversaries $\mathcal{A}$, it is sufficient to consider only the most simple adversary possible. The Dummy Adversary $\mathcal{D}$ takes all messages it receives from the environment $\mathcal{E}$ and forwards them without any change to the concerned protocol party. The other way around, if the $\mathcal{D}$ receives messages from any protocol party, it forwards them directly to the environment.

This sounds contradictory on the first glance, as we restrict ourselves to a very special adversary. However, the intuition for this fact is simple. The goal of the environment is to distinguish whether it is "talking" to the adversary and a real protocol execution of π or to the simulator and the ideal protocol execution with $\mathcal{F}$. Now if $\mathcal{A}$ would not forward a message to $\mathcal{E}$, that can only make $\mathcal{E}$'s task harder as its view of the interaction is "not complete". Analogously, if the adversary interacts with any party without the environment knowing, or if the adversary does not interact with a party even though $\mathcal{E}$ instructed him to do so, this does only make the task harder for $\mathcal{E}$. Canetti [Can00, Claim 11] proves this formally.

2.5. Oblivious Transfer

Oblivious Transfer (OT), introduced in 1981 by Rabin [Rab05], is fundamental for many cryptographic tasks, including Yaos's garbled circuits. OT allows a sender to transfer one of two messages to a receiver. The receiver can choose whether it wants the first or the second message. The security guarantee for the sender is that the receiver does not learn anything about the message that was not chosen. The security guarantee for the receiver is that the sender does not learn anything about the choice of the receiver. This is sketched graphically in Figure 2.4. The above describe protocol is also known as 1-out-of-2 OT. In more generality, one can define a 1-out-of-m OT, where the sender sends $m \in \mathbb{N}$ messages $x_1, \ldots, x_m$. One can also distinguish between “bit-OT”, where the messages of the sender are single bits $x_0, x_1 \in \{0, 1\}$ and “string-OT”, where the messages are bit strings length $x_0, x_1 \in bitstr$. In the case of garbled circuits, OT allows the garbling party to provide the evaluating party with the necessary input labels without learning which input bits the evaluating party actually chooses. That means, we need to perform $n \in \mathbb{N}$ executions of a 1-out-of-2 string-OT protocol, one for each of the n input bits of the circuit. As we are interested in universally composable OTs, we briefly recall two ideal functionalities for UC-secure OT and discuss their differences before we define the functionality we will use in this work.

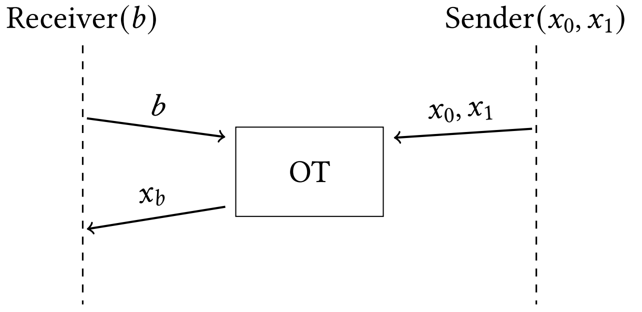


Figure 2.4.: Sketch of the 1-Out-of-2 OT Functionality.

The first functionality $\mathcal{F}'_{\mathrm{OT}}$ is introduced by Canetti et al. [Can+02] and used e.g. by Peikert, Vaikuntanathan, and Waters [PVW08]. It considers *one* sender and *one* receiver. We present the functionality (for the special case of 1-out-of-2 OT) in Figure 2.5. With this functionality, each sender-receiver pair requires its own CRS. Additionally, this functionality does not inform the adversary explicitly if the messages are sent to functionality.

The second functionality $\mathcal{F}_{\mathrm{MOT}}$ is used e.g. by Choi et al. [Cho+13]. They claim that it was introduced by Canetti [Can00], however we couldn't find a version of [Can00], where this functionality appeared. Thus, we present the ideal functionality $\mathcal{F}_{\mathrm{MOT}}$ for multi-session OT as defined by [Cho+13] in Figure 2.6. In contrast to the first functionality, the adversary is explicitly informed, whenever a message is sent to the functionality. This is done by $(\mathsf{Send}, \langle sid, ssid, P_i, P_j \rangle)$ messages from the ideal functionality to the adversary. A second difference seems to be the ability of the adversary to delay messages. Concretely, the result of the OT is only transferred to the receiver P_j after the adversary sent $(\mathsf{Received}, \langle sid, ssid, P_i, P_j \rangle, P_j)$ to the functionality. However, in the UC-Framework the adversary is always able to delay messages. Thus, this messages is more a syntactical difference in the two functionalities. The last difference is the number of parties. The second functionality in Figure 2.6 allows for up to n parties. This might seem more suited for our case on the first glance. By employing the second functionality, one does only need to generate one CRS for all OT executions. But the subtle problem that occurs is that the number of parties n needs to be fixed at the beginning of the protocol execution. In contrast, in our OPRF scenario we allow (honest) parties to join the protocol at any time. If we would want to use $\mathcal{F}_{\text{MOT}}$ in such a dynamic context we would need to instantiate a new protocol instance –and thus generate a new CRS– whenever more that n parties join the execution. To avoid those problems, we rather opt to use a "single sender – single receiver" functionality, like Figure 2.5. Note, that the generation of a new CRS for every OT execution is not problematic as we are in the Ranom Oracle Model (ROM). We can just generate a CRS for the OT execution by "hashing" the session id and the prefix of the protocol execution using the random oracle.

For the sake of clarity, we augment the original $\mathcal{F}'_{\text{OT}}$ functionality from [Can00] with explicit message allowing the sender to delete execution. We stress again that this does not give the adversary additional power but rather makes properties of the UC-framework more explicit. We describe our functionality $\mathcal{F}_{\text{OT}}$ in Figure 2.7.

Functionality $\mathcal{F}'_{\text{OT}}$

$\mathcal{F}'_{\text{OT}}$ proceeds as follows, interaction and running with an oblivious transfer sender T, a receiver R and an adversary S.

- Upon receiving a message $(\mathsf{Sender}, sid, x_0, x_1)$ from T, where each $x_j \in \{0,1\}^m$, record (x_0, x_1). (The lengths of the strings m is fixed and known to all parties.)
- Upon receiving a message $(\mathsf{Receiver}, sid, i)$ from R, where $i \in \{0,1\}$ send (sid, x_i) to T and sid to S and halt. If no $(\mathsf{Sender}, \dots)$ message was sent, then send nothing to R.

Figure 2.5.: The Ideal Functionality $\mathcal{F}'_{\text{OT}}$ From [Can+02].

2.6. Garbled Circuits

2.6.1. Boolean Circuits

A boolean circuit is model of computation like the Turing machine. They can be seen as a mathematical abstraction of actual electrical circuits that are used to build processors. We define them as follows:

Definition 8 (Boolean Circuit) [AB09, Def. 6.1], [BHR12, Sec. 2.3] For $n, m \in \mathbb{N}$, a *boolean circuit C* with n inputs and m outputs is a directed acyclic graph. It contains n nodes with no incoming edges; called the input nodes and m nodes with no outgoing edges, called the output nodes. All other nodes are called gates. In our case, they are labeled with either

Functionality $\mathcal{F}_{\text{MOT}}$

$\mathcal{F}_{\text{MOT}}$ interacts with parties $P_1, \ldots, P_n$ and an adversary Sim and proceeds as follows:

- Upon receiving a message $(\mathsf{Send}, \langle sid, ssid, P_i, P_j \rangle, \langle x_0, x_1 \rangle)$ from P_i, where each $x_j \in \{0,1\}^m$, record $\langle ssid, P_i, P_j, x_0, x_1 \rangle$. Reveal $(\mathsf{Send}, \langle sid, ssid, P_i, P_j \rangle)$ to the adversary. Ignore further $(\mathsf{Send}, \ldots)$ messages from P_i with the same $ssid$.
- Upon receiving a message $(\mathsf{Receive}, \langle sid, ssid, P_i, P_j \rangle, b)$ from P_j, where $b \in \{0,1\}$ record the tuple $\langle ssid, P_i, P_j, b \rangle$ and reveal $(\mathsf{Receive}, \langle sid, ssid, P_i, P_j \rangle)$ to the adversary. Ignore further $(\mathsf{Receive}, \ldots)$ messages from P_j with the same $ssid$.
- Upon receiving a message $(\mathsf{Sent}, \langle sid, ssid, P_i, P_j \rangle, P_i)$ from the adversary, ignore the message if $\langle ssid, P_i, P_j, x_0, x_1 \rangle$ or $\langle ssid, P_i, P_j, b \rangle$ is not recorded; Otherwise return $(\mathsf{Sent}, \langle sid, ssid, P_i, P_j \rangle)$ to P_i; Ignore further $(\mathsf{Sent}, \langle sid, ssid, P_i, P_j \rangle, P_i)$ messages from the adversary.
- Upon receiving a message $(\mathsf{Received}, \langle sid, ssid, P_i, P_j \rangle, P_j)$ from the adversary ignore the message if $\langle ssid, P_i, P_j, x_0, x_1 \rangle$ or $\langle ssid, P_i, P_j, b \rangle$ is not recorded; Otherwise return $(\mathsf{Received}, \langle sid, ssid, P_i, P_j \rangle, m_b)$ to P_j; Ignore further $(\mathsf{Received}, \langle sid, ssid, P_i, P_j \rangle, P_j)$ messages from the adversary.

Figure 2.6.: The Ideal Functionality $\mathcal{F}_{\text{MOT}}$ From [Cho+13].

XOR or *AND*. For a gate g, $G(g) \in \{XOR, AND\}$ yields the function corresponding to the label. Gates have always two inputs and arbitrary fan-out. □

Further, [BHR12] use the convention that all wires of the circuit are numbered. If $q \in \mathbb{N}$ is the number of gates then $r = n + q$ is the number of wires. The number of the outgoing wire(s) of a gate serves as number of the gate. Further, they assume that the numbering is ordered in the following sense. Let $A : \text{Gates} \longrightarrow \text{Wires}$ give the first input wire of a gate and $B : \text{Gates} \longrightarrow \text{Wires}$ give the second input wire of a gate. Then it holds that for all $g \in \text{Gates}$ that $A(g) < B(g) < g$. By using this convention, the evaluation of the circuit can be defined as follows:

Definition 9 (Circuit Evaluation) A boolean circuit is evaluated by iterating over all gates $g \in \text{Gates}$ in their order and setting $a := A(g)$, $b := A(g)$, $x_g := G(g)(x_a, x_b)$. The output of the circuit is $x_{n+q-m+1} \parallel \ldots \parallel x_{n+q}$. □

Note that x_a and x_b are well-defined, as the circuit is ordered.

2.6.2. Yao's Garbled Circuits

Garbled Circuits were introduced by Andrew Yao. According to [BHR12], the original idea stems from an oral presentation of [Yao86]. Later, several works as e.g. Goldreich, Micali, and Wigderson [GMW87] and Beaver, Micali, and Rogaway [BMR90] described the protocol in more detail. Since their emergence, they went from a purely theoretical

Functionality $\mathcal{F}_{\mathrm{OT}}$

$\mathcal{F}_{\mathrm{OT}}$ proceeds as follows, interaction and running with an oblivious transfer sender S, a receiver R and an adversary $\mathcal{A}$.

- Upon receiving a message $(\text{OT-Send}, sid, (x_0, x_1))$ from S, where each $x_j \in \{0,1\}^m$, record $\langle sid, x_0, x_1 \rangle$. Reveal $(\text{OT-Send}, sid)$ to the adversary. Ignore further $(\text{OT-Send}, \dots)$ messages from S with the same sid.
- Upon receiving a message $(\text{OT-Receive}, sid, b)$ from R, where $b \in \{0,1\}$ record the tuple $\langle sid, b \rangle$ and reveal $(\text{OT-Receive}, sid)$ to the adversary. Ignore further $(\text{OT-Receive}, \dots)$ messages from R with the same sid.
- Upon receiving a message $(\text{OT-Sent}, sid)$ from the adversary, ignore the message if $\langle sid, x_0, x_1 \rangle$ or $\langle sid, b \rangle$ is not recorded; Otherwise return $(\text{OT-Sent}, sid)$ to S; Ignore further $(\text{OT-Sent}, sid, \dots)$ messages from the adversary.
- Upon receiving a message $(\text{OT-Received}, sid)$ from the adversary ignore the message if $\langle sid, x_0, x_1 \rangle$ or $\langle sid, b \rangle$ is not recorded; Otherwise return $(\text{OT-Received}, sid, x_b)$ to R; Ignore further $(\text{OT-Received}, sid, \dots)$ messages from the adversary.

Figure 2.7.: Our Ideal Functionality $\mathcal{F}_{\mathrm{OT}}$.

construct to a practically interesting and powerful cryptographic tool. Garbled Circuits allow two parties, Alice and Bob, to jointly evaluate a boolean circuit. The circuit takes a secret input from Alice and a secret input from Bob. After the execution, both parties (or only one of them) learns the output.

From a abstract point of view, the protocol works as follows: Alice encodes her input and the boolean circuit in a way, such that Bob can evaluate the circuit on the encoded input, but learns nothing about the input. Alice sends the encoded input and circuit to Bob. Then she encodes the possible input bits for Bob. Bob uses an OT protocol to get his encoded input from Alice, while Alice learns nothing about the input of Bob. Bob then evaluates the circuit and gets an encoded output. Both parties get the result by decoding this output.

More precisely, Alice "garbles" the boolean circuit. This means, she assigns random bit strings of length proportional to the security parameter for each possible input bit. These so-called labels hide the actual inputs. Next, she encrypts for every gate of the boolean circuit the output of the gate with the corresponding input labels as keys. This means, she performs one encryption for each row of the truth table of the gate. Finally, she permutes the order of the rows, so the order of the ciphertexts does not reveal information on the outcome of the gate. After that, she sends the garbled circuit to Bob, together with the input labels for her input. Then, Alice and Bob perform a 1-out-of-2 OT for each input bit of Bob. With the OTs, Bob gets the labels for his input bits, while Alice learns nothing about his input. Now Bob can evaluate the circuit gate by gate, as he has the garbled circuit and both sets of input labels. Again, he proceeds gate by gate. He tries to decrypt the output label of a gate by using the input labels. For one row, the decryption will work and Bob receives the output label of the gate. In the textbook-version, the encryption must ensure that decrypting with wrong keys can be detected by Bob. Eventually, Bob gets the output labels from evaluating the output gates. Now, he can e.g. send the output labels back to Alice. Alice knows the mapping of the output labels to actual output values and thus, learns the result of the computation. This is depicted in Figure 2.8.

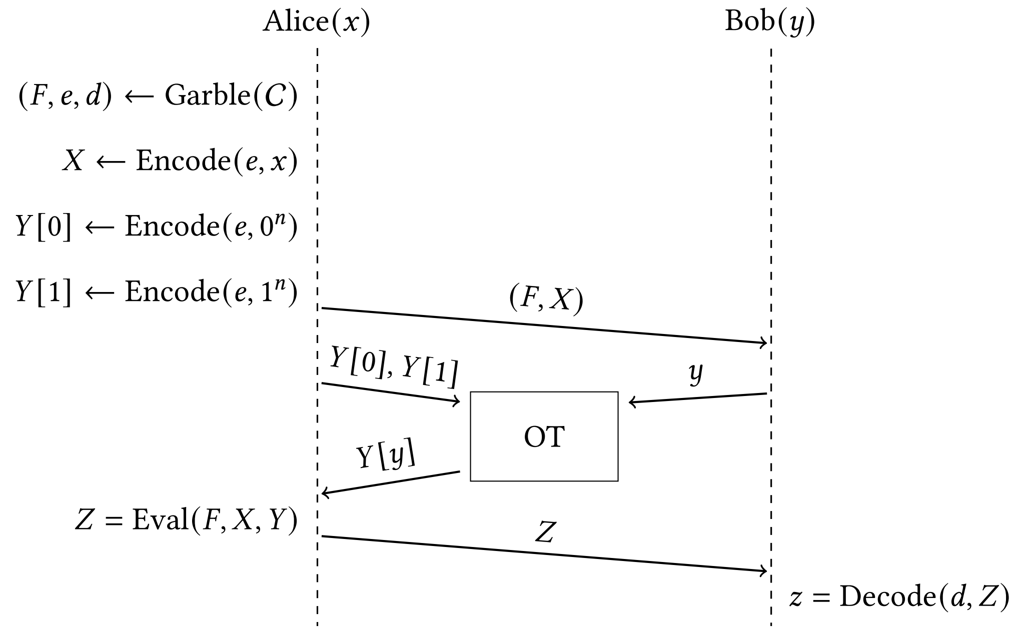

Alice(x)
Bob(y)
$(F, e, d) \leftarrow \text{Garble}(C)$
$X \leftarrow \text{Encode}(e, x)$
$Y[0] \leftarrow \text{Encode}(e, 0^n)$
$Y[1] \leftarrow \text{Encode}(e, 1^n)$
(F, X)
$Y[0], Y[1]$
y
OT
$Y[y]$
$Z = \text{Eval}(F, X, Y)$
Z
$z = \text{Decode}(d, Z)$

Figure 2.8.: The Garbled Circuit Protocol.

The original construction offers only passive security. This means that the protocol is secure as long as both parties follow the protocol description but try to gain additional information. If both parties follow the protocol, they cannot learn anything more from the transaction, than the output. Obviously, this holds no longer, if the garbling party Alice deviates from the protocol. The garbler could for example simply garble a different circuit, even one that leaks information on the evaluators input.

There are several possibilities to transform a passively secure garbled circuit protocol into an actively secure one. The most common technique is called *cut-and-choose*. It was first used in the context of blind signatures by Chaum [Cha83] and was later adapted to the garbled circuit setting by Mohassel and Franklin [MF06] and Lindell and Pinkas [LP07]. The idea is to garble the circuit many times with independent randomness. The evaluator then randomly challenges the garbler to "reveale" some of the circuits, i.e., showing the used randomness in the generation, to prove that the garbling was done correctly. The evaluator accepts the garbling only if all of the checks were successful. Wang, Ranellucci, and Katz [WRK17] introduced a technique called *authenticated garbling*. The approach is to combine authenticated secret sharing (i.e. Bendlin et al. [Ben+11]) with garbled circuits to achieve active security. Finally, Goldreich, Micali, and Wigderson [GMW87] introduced a generic approach to transform any passively secure protocol into an actively secure one. However, as this approach is very generic, it is likely too inefficient for our purposes.

Several works improved the efficiency of garbled circuits. First, Beaver, Micali, and Rogaway [BMR90] introduced the point-and-permute technique. By assigning two additional bits to a ciphertext, the evaluator can directly identify the entry in the truth table, which has to be decrypted. Therefore, the number of decryptions per gates is reduced from four to one, as the evaluator does not have to try decrypting every row of the truth table. Naor, Pinkas, and Sumner [NPS99] and Pinkas et al. [Pin+09] further reduced the number of necessary encryptions to garble the circuit. The most notable advances are the techniques called *free-xor* and *half-gates*. Free-xor was introduced by Kolesnikov and Schneider [KS08] and allows to garble a circuit in such a way, that xor-gates cost no additional encryption. This is particularly useful as common circuits, like e.g. Advanced Encryption Standard (AES) contain a much bigger number of xor-operations than and-operations. Half-gates [ZRE15] reduces the number of encryptions needed to encode an and-gate from four encryptions to only two. Recently, Rosulek and Roy [RR21] even enhanced the half-gates technique, circumventing the lower bound proven in [ZRE15]. They introduce a technique called *slicing and dicing*. With that technique xor-gates are still free and and-gates cost $1.5\lambda + 5$ bits per gate, where λ is the security parameter. Free-xor and half-gates can be combined to offer very efficient garbling.

Over the years, several frameworks were developed to facilitate the real-world implementation of garbled circuits. The first implementation was the *Fairplay* library [Mal+04]. As this library is relatively old, it is merely of historical interest. Other libraries like Kreuter, shelat, and Shen [KsS12] do not feature the optimizations provided by half-gates [ZRE15]. Therefore we will use the *emp-toolkit* library by Wang, Malozemoff, and Katz [WMK16] in this work. This C++ library offers a method to implement garbled circuits that are passively and actively secure. Additionally, most recent optimizations like free-xor and half-gates are implemented. The downside is that the framework is barely documented.

2.6.3. Garbling Schemes

Bellare, Hoang, and Rogaway [BHR12] defined an elegant abstraction of the above described protocol and gave a thorough analysis of security properties offered by variants of these algorithms. In their work, the use the side-information function Φ. Given a circuit f, this functions outputs certain information about the circuit. Depending on the desired level of security, one can define Φ differently. E.g. $\Phi(f) = f$ would mean that all parties learn the whole description of the circuit. In a more restrictive setting, one could also demand e.g. that $\Phi(f) = n$, where n is the number of input bits of f. However, in this work we will always assume the first case, i.e. that the circuit description is public. We render their definition here:

Definition 10 (Garbling Scheme) [BHR12, Sec. 3.1] A *garbling scheme* is a tuple $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, where Gb is probabilistic and the remaining algorithms are deterministic. Let $f \in \{0,1\}^*$ be a description of the function we want to garble. The function $\mathsf{ev}(f, \cdot) : \{0,1\}^n \to \{0,1\}^m$ denotes the actual function, we want to garble. (n and m must be efficiently computable from f.) On input f and a security parameter $\lambda \in \mathbb{N}$, algorithm Gb returns a triple of strings $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f)$. String e describes an encoding function, $\mathsf{En}(e, \cdot)$, that maps an initial input $x \in \{0,1\}^n$ to a garbled input $X = \mathsf{En}(e, x)$. String F

describes a garbled function, $\mathsf{Ev}(F, \cdot)$, that maps each garbled input X to a garbled output $Y = \mathsf{Ev}(F, X)$. String d describes a decoding function, $\mathrm{De}(d, \cdot)$, that maps a garbled output Y to a final output $y = \mathrm{De}(d, Y)$. □

The security properties defined in [BHR12] are *privacy*, *obliviousness*, and *authenticity*. Intuitively, privacy means that no efficient adversary can calculate anything from the garbled circuit F, the input labels X and the decoding information d that the adversary could not have calculated from the output value y and the side-information $\Phi(f)$ alone. In particular, the adversary cannot "break" the garbling scheme to get the input value of one of the parties. *Obliviousness* is a related notion. The intuition is that no efficient adversary can calculate anything from the garbled circuit F and the input labels X that the adversary could have calculated from the side-information $\Phi(f)$.

Note here, that in contrast to *privacy*, the adversary is not given the decoding information d. Consequently, the adversary should not be able to produce an output just from the garbled circuit F and the input labels X. This is reflected in the fact that the simulator in the security experiment Figure 2.10 will not get the output value $y \leftarrow \mathrm{ev}(f, x)$. The last notion is *authenticity*. The idea behind this notion is that the only output one should be able to produce using the garbled circuit is $y = \mathrm{De}(d, \mathsf{Ev}(F, X))$.

Bellare, Hoang, and Rogaway [BHR12] gave a game-based security definition, as well as a simulation based security definition for the first two properties. As Zahur, Rosulek, and Evans [ZRE15] use the simulation-based notions, we will only render the simulation-based definitions and the definition of authenticity here.

Definition 11 (Privacy) [BHR12, Sec. 3.4] For a simultor $\mathcal{S}$, we define the advantage of adversary $\mathcal{A}$ in the security experiment defined in Figure 2.9, as

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{prv.sim},\Phi,\mathcal{S}}(\mathcal{A}, \lambda) := 2\Pr[\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}} = 1] - 1.$$

A garbling scheme has *privacy* if for every PPT adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{prv.sim},\Phi,\mathcal{S}}(\mathcal{A}, \lambda) \leq \mathrm{negl}(\lambda),$$

for a negligible function $\mathrm{negl}(\cdot)$. □

Definition 12 (Obliviousness) [BHR12, Sec. 3.5] For a simulator $\mathcal{S}$, we define the advantage of adversary $\mathcal{A}$ in the security experiment defined in Figure 2.10, as

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{obv.sim},\Phi,\mathcal{S}}(\mathcal{A}, \lambda) := 2\Pr[\mathrm{ObvSim}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}} = 1] - 1.$$

A garbling scheme has *obliviousness* if for every PPT adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{obv.sim},\Phi,\mathcal{S}}(\mathcal{A}, \lambda) \leq \mathrm{negl}(\lambda),$$

for a negligible function $\mathrm{negl}(\cdot)$. □

Game $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$

- The challenger C chooses a bit $b \in \{0,1\}$ uniformly at random.
- $\mathcal{A}$ sends a function $f : \{0,1\}^n \longrightarrow \{0,1\}^*$ and input $x \in \{0,1\}^n$ to C.
- If $x \notin \{0,1\}^n$ the challenger sends $\perp$ to $\mathcal{A}$.
  Else if $b = 1$ the challenger sets $(F, e, d) \leftarrow \mathrm{Gb}(1^\lambda, f)$ and $X \leftarrow \mathsf{En}(e, X)$.
  Else the challenger calculates $y \leftarrow \mathrm{ev}(f, x)$ and simulates $(F, X, d) \leftarrow \mathcal{S}(1^\lambda, y, \Phi(f))$.
  Finally, C sends (F, X, d) to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b'
- The game outputs $b' \overset{?}{=} b$.

Figure 2.9.: The Simulation-Based Privacy Game From [BHR12, Fig. 5].

Definition 13 (Authenticity) [BHR12, Sec. 3.6] We define the advantage of adversary $\mathcal{A}$ in the security experiment defined in Figure 2.11, as

$$\mathrm{Adv}^{\mathsf{aut}}_{\mathcal{G}}(\mathcal{A}, \lambda) := \Pr[\mathrm{Aut}^{\mathcal{A}}_{\mathcal{G}} = 1].$$

A garbling scheme has *authenticity* if for every PPT adversary $\mathcal{A}$ it holds that

$$\mathrm{Adv}^{\mathsf{aut}}_{\mathcal{G}}(\mathcal{A}, \lambda) \leq \mathrm{negl}(\lambda),$$

for a negligible function $\mathrm{negl}(\cdot)$. □

2.6.4. Free-Xor

The technique proposed in Kolesnikov and Schneider [KS08] is one of the most important advances on efficiency of garbled circuits. The technique allows to calculate a garbled circuit in such a way that xor-gates come with no additional data, i.e., encrypted gate labels that have to be sent over the network. The gist of [KS08] is the following: If one defines the input labels of a xor-gate as $X[1] = X[0] \oplus \Delta$ and $Y[1] = Y[0] \oplus \Delta$, where Δ is some secret constant known to the garbling party and $X[0], Y[0]$ are random labels and one defines $Z[0] = X[0] \oplus Y[0]$, the evaluating party can calculate the output of the xor-gate *locally*. This is because we have for $b_1, b_2 \in \{0,1\}$

$$X[b_1] \oplus Y[b_2] = Z[b_1 \oplus b_2],$$

and the evaluating party can compute the xor of the two input labels without further information from the garbling party. Kolesnikov and Schneider [KS08] argue that even though the labels are not chosen independently anymore in the above case, the garbling is still secure. This technique is particularly useful as some the circuit description of some "real-world" functions contain a relatively high amount of xor-gates. For example, AES can be realized with 28216 gates from which 55% are xor-gates [Pin+09].

Game $\mathrm{ObvSim}_{\mathcal{G},\Phi,\mathcal{S}}$

- The challenger C chooses a bit $b \in \{0,1\}$ uniformly at random.
- $\mathcal{A}$ sends a function $f : \{0,1\}^n \longrightarrow \{0,1\}^*$ and input $x \in \{0,1\}^n$ to C.
- If $x \notin \{0,1\}^n$ the challenger sends $\perp$ to $\mathcal{A}$.
  Else if $b = 1$ the challenger sets $(F, e, d) \longleftarrow \mathsf{Gb}(1^\lambda, f)$ and $X \longleftarrow \mathsf{En}(e, X)$.
  Else the challenger simulates $(F, X) \longleftarrow \mathcal{S}(1^\lambda, \Phi(f))$.
  Finally, C sends (F, X) to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b'
- The game outputs $b' \overset{?}{=} b$.

Figure 2.10.: The Simulation-Based Obliviousness Game From [BHR12, Fig. 5].

Game $\mathrm{Aut}_{\mathcal{G}}$

- $\mathcal{A}$ sends a function $f : \{0,1\}^n \longrightarrow \{0,1\}^*$ and input $x \in \{0,1\}^n$ to C.
- If $x \notin \{0,1\}^n$ the challenger sends $\perp$ to $\mathcal{A}$.
  The challenger sets $(F, e, d) \longleftarrow \mathsf{Gb}(1^\lambda, f)$ and $X \longleftarrow \mathsf{En}(e, X)$.
  C sends (F, X) to $\mathcal{A}$.
- $\mathcal{A}$ sends Y to C
- The game outputs 1 iff $(\mathsf{De}(d, Y) \neq \perp$ and $Y \neq \mathsf{Ev}(F, X)$.

Figure 2.11.: The Authenticity Game From [BHR12, Fig. 5].

2.6.5. Half-Gates

Zahur, Rosulek, and Evans [ZRE15] proposed an optimization to Yao's garbled circuits that reduces the cost of each and-gate by half. This huge improvement is particularly interesting as it can be combined with the free-xor optimization technique. The key idea is to split a single and-gate into two "half-gates" that are easier to handle.

To understand the technique, one has to consider the following. Let's assume we want to garble a gate $c = a \wedge b$. We further assume that the free-xor technique as in Section 2.6.4 is used. Then we have labels C, $C \oplus R$, A, $A \oplus R$, B, and $B \oplus R$ for this gate, where R is the free-xor offset and A, B, C, are the labels encoding zero. If we assume that the garbler (somehow) already knows the value of a, it would be easy to garble the gate. If $a = 0$ the garbler could just garble a gate that outputs constant 0 and for $a = 1$ the garbler could garble an "identitiy" gate, i.e., a gate that always outputs b. So the garbler has to produce two encryptions:

$$\begin{aligned} & H(B) \oplus C \\ \textbf{if } a = 0\colon\ & H(B \oplus R) \oplus C \\ \textbf{if } a = 1\colon\ & H(B \oplus R) \oplus C \oplus R \end{aligned}$$

This is the first "half-gate". For the second "half-gate", we adapt this idea to the evaluator side. Consider again an and-gate $c = a \wedge b$. But this time, we assume that the evaluator (somehow) already knows the bit a. If the evaluator knows the value of a, it can behave differently in evaluating the circuit. For $a = 0$, the evaluator has to receive the label C, as the output is always zero. If $a = 1$, the output of the gate depends on the value of b. It is sufficient for the evaluator to learn the label $\Delta := C \oplus B$. By adding either B or $B \oplus R$ to Δ, the evaluator will receive the right output label, i.e., either C or $C \oplus R$. This means, the "half gate" of the evaluator is comprised of two encryptions

$$\begin{aligned} & H(A) \oplus C \\ & H(A \oplus R) \oplus C \oplus B. \end{aligned}$$

One can further use optimization from garbled-row-reduction [NPS99] to reduce the number of encryptions for each of the "half-gates" to just one. To finally put those two halves together, we use the fact that for any $r \in \{0, 1\}$ we have

$$\begin{aligned} c &= a \wedge b \\ &= a \wedge ((r \oplus r)b) \\ &= (a \wedge r) \oplus (a \wedge (r \oplus b)). \end{aligned}$$

If we let the garbler choose a uniformly random value $r \in \{0, 1\}$, we can regard the and-gate $(a \wedge r)$ as the garbler's "half-gate". Obviously, r is known to the garbler. We can further regard $(a \wedge (r \oplus b))$ as the evaluator's "half-gate", if we can transfer the value of $(r \oplus b)$ to the evaluator. This can be done via the choice bit of the point-and-permute technique, see [BMR90]. Intuitively, $(r \oplus b)$ does not leak any information about b to the evaluator, as b is masked by the uniformly random value r. The xor-gate that is used to combine the two halves is free.

We recall the details in Figure 2.12. For this figure, we adhered to the notation of [ZRE15] and denote by $\hat{x}$ the vector $(x_0, \ldots, x_n)$, for some $n \in \mathbb{N}$. Further, NextIndex is a stateful procedure that simply increments an internal counter. Zahur, Rosulek, and Evans [ZRE15] show that their scheme satisfies the simulation-based notions of obliviousness and privacy, see Section 2.6.3.

2.7. Security of OPRFs

2.7.1. Simulation-Based Security

Freedman et al. [Fre+05] defined the security of OPRF using the real-world/ideal-world paradigm. They define two notions of OPRF, namely *strong-OPRF* and *relaxed-OPRF* (later also called *weak OPRF*). The first definition requires, that the user learns nothing about the server's key. Though this is the intuitive property that we want from an OPRF, this definition is to strong to capture some efficient protocols. For example, if the user receives a value from the server and finally applies a hash function to that value to obtain the final PRF output, the client obviously learned more about the server's that just the PRF output. It learned the hash-preimage of the PRF output. E.g. the constructions from Jarecki, Krawczyk, and Xu [JKX18] and Jarecki et al. [Jar+16] or Kolesnikov et al. [Kol+16] do not satisfy the strong-OPRF notion because of their application of a hash function. Thus, [Fre+05] define a relaxed version of OPRF. [Fre+05] give a brief definition of relaxed-OPRF. We work out the details in the following:

Definition 14 (Relaxed-OPRF) [Fre+05, Def. 6] A two party protocol π between a user U and a server S is said to be a *relaxed-OPRF* if there exists some PRF familiy f_k, such that π correctly realizes the following functionality:

- Inputs: User holds an input $x \in \mathcal{X}$ and server a key $k \in \mathcal{K}$,
- Output: User outputs $f_k(x)$ and server outputs nothing,

and if the following properties hold:

- *User privacy*: There exists a PPT machine Sim such that for every key $k \in \mathcal{K}$ and every input $x \in \mathcal{X}$ it holds that
$$\{v \mid v = \mathsf{view}_{\mathsf{S}}\langle \mathsf{S}(k), \mathsf{U}(x)\rangle_\pi\}_\lambda \overset{c}{\approx} \{v \mid v \leftarrow \mathsf{Sim}(1^\lambda, k)\}_\lambda.$$
- *Server privacy*: We demand that for any malicious PPT adversary $\mathcal{A}$ playing the role of the client there exists a PPT simulator Sim such that for all inputs $((x_1, x_2, \ldots, x_n), w)$ it holds that
$$\{(v, f_k(x_1), f_k(x_2), \ldots, f_k(x_n) \mid k \xleftarrow{\$} \mathcal{K}, v = \mathsf{out}_{\mathcal{A}}\langle \mathsf{S}(k), \mathcal{A}(w)\rangle\}$$
$$\overset{c}{\approx} \{(\mathsf{Sim}(f_k(w)), f_k(x_1), f_k(x_2), \ldots, f_k(x_n)) \mid k \xleftarrow{\$} \mathcal{K}\},$$
where S is a honest server and $\mathsf{view}_{\mathsf{P}}\langle \mathsf{A}(x), \mathsf{B}(y)\rangle_\pi$ denotes the view of party $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ when protocol π is executed between A with input x and party B with input y and $\mathsf{out}_{\mathsf{P}}\langle \mathsf{A}(x), \mathsf{B}(y)\rangle_\pi$ denotes the output of that interaction. Ist das wirklich richtig? Liest sich komisch. □

```
$\mathsf{Gb}(1^\lambda, f)$ :
$R \xleftarrow{\$} \{0,1\}^{\lambda-1} \parallel 1$
**for** $i$ **in** $\mathsf{Inputs}(f)$ **do**
  $W_i^0 \xleftarrow{\$} \{0,1\}^\lambda$
  $W_i^1 \coloneqq W_i^0 \oplus R$
  $e_i \coloneqq W_i^0$
// In topological order
**for** $i \notin \mathsf{Inputs}(f)$ **do**
  $\{a, b\} \coloneqq \mathsf{GateInputs}(f, i)$
  **if** $i \in \mathsf{XorGates}(f)$:
    $W_i^0 \coloneqq W_a^0 \oplus W_b^0$
  **else**
    $(W_i^0, T_{Gi}, T_{Ei}) \coloneqq \mathsf{GbAnd}(W_a^0, W_b^0)$
    $F_i \coloneqq (T_{Gi}, T_{Ei})$
  **endif**
  $W_i^1 \coloneqq W_i^0 \oplus R$
**for** $i \in \mathsf{Outputs}(f)$ **do**
  $d_i \coloneqq \mathsf{lsb}(W_i^0)$
**return** $(\hat{F}, \hat{e}, \hat{d})$
```

```
private $\mathsf{GbAnd}(W_a^0, W_b^0)$ :
$p_a \coloneqq \mathsf{lsb}(W_a^0), p_b \coloneqq \mathsf{lsb}(W_b^0)$
$j \coloneqq \mathsf{NextIndex}(), j' \coloneqq \mathsf{NextIndex}()$
// First Half-Gate
$T_G \xleftarrow{\$} H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
$W_G^0 \xleftarrow{\$} H(W_a^0) \oplus p_a T_G$
// Second Half-Gate
$T_E \xleftarrow{\$} H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
$W_E^0 \xleftarrow{\$} H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
// combine halves
$W^0 \coloneqq W_G^0 \oplus W_E^0$
**return** $(W^0, T_E, T_G)$
```

```
$\mathsf{Ev}(\hat{F}, \hat{X})$ :
**for** $i \in \mathsf{Inputs}(\hat{F})$ **do**
  $W_i \coloneqq X_i$
// In topological order
**for** $i \notin \mathsf{Inputs}(\hat{F})$ **do**
  $\{a, b\} \coloneqq \mathsf{GateInputs}(\hat{F}, i)$
  **if** $i \in \mathsf{XorGates}(\hat{F})$:
    $W_i \coloneqq W_a \oplus W_b$
  **else**
    $s_a \coloneqq \mathsf{lsb}(W_a), s_b \coloneqq \mathsf{lsb}(W_b)$
    $j \coloneqq \mathsf{NextIndex}()$
    $j' \coloneqq \mathsf{NextIndex}()$
    $(T_{Gi}, T_{Ei}) \coloneqq F_i$
    $W_{Gi} \xleftarrow{\$} H(W_a, j) \oplus s_a T_{Gi}$
    $W_{Ei} \xleftarrow{\$} H(W_b, j') \oplus s_b(T_{Ei} \oplus W_a)$
    $W_i \coloneqq W_{Gi} \oplus W_{Ei}$
  **endif**
**for** $i \in \mathsf{Outputs}(\hat{F})$ **do**
  $Y_i \coloneqq W_i$
**return** $\hat{Y}$
```

```
$\mathsf{En}(\hat{e}, \hat{x})$ :
**for** $e_i \in \hat{e}$ **do**
  $X_i \coloneqq e_i \oplus x_i R$
**return** $\hat{X}$
```

```
$\mathsf{De}(\hat{d}, \hat{Y})$ :
**for** $d_i \in \hat{d}$ **do**
  $y_i \coloneqq d_i \oplus \mathsf{lsb}(Y_i)$
**return** $\hat{y}$
```

Figure 2.12.: The Procedures for Garbling a Function f.

Functionality $\mathcal{F}_{\text{AUTH}}$

- Upon invocation, with input $(\textsc{Send}, mid, R, m)$ from S, send backdoor message $(\textsc{Sent}, mid, S, R, m)$ to the adversary.
- Upon receiving backdoor message $(\textsc{ok}, mid)$: If not yet generated output, then output $(\textsc{Sent}, mid, S, R, m)$ to R.

Figure 2.13.: The Ideal Functionality $\mathcal{F}_{\text{AUTH}}$ From [Can00].

Functionality $\mathcal{F}_{\text{RO}}$

Upon receipt of a message $x \in A$, if there is a record $\langle x, y \rangle$, return y. Else draw $y^* \in B$ uniformly at random. Record $\langle x, y^* \rangle$ and return y^*.

Figure 2.14.: The Ideal Functionality $\mathcal{F}_{\text{RO}}$.

2.7.2. Universally Composable OPRFs

2.7.2.1. Authenticated Channels

In this work we will use the notion of authenticated channels. Intuitively, this means that a sender of a message can be sure that only the intended receiver (or no one, in case the message is lost) can receive a message. Additionally, a sender can be sure that the message was not altered by an adversary. We demand that those requirements do only hold as long as both parties follow the protocol.

Canetti [Can00] define authenticated communication via the ideal functionality depicted in Figure 2.13.

2.7.2.2. The UC Framework and Random Oracles

A random oracle is an (over-) idealization of a hash function. Assuming the existence of a random oracle often allows to prove security of cryptographic objects, that are are more efficient than their “plain-model” counterparts. In a real-world implementation, the random oracle will be replaced by a cryptographic hash function. While there are examples, where this replacement does not preserve security (see [CGH98]), the random oracle model is still regarded as a useful heuristic. A random oracle $H : A \to B$ maps elements from set A to elements of set B. It can be queried by all parties. If the random oracle receives an input query $x \in A$ for the first time, it draws a uniformly random output value $y \in B$ and outputs this value. The oracle also stores the tuple $\langle x, y \rangle$. If the random oracle receives the query x again, it outputs y and does not draw a new value.

In the UC-framework, the random oracle is modeled as an ideal functionality. We describe such a functionality in Figure 2.14. However, for the sake of convenience, we will notate the random oracle in our work like a “conventional hash function” and not like an ideal functionality.

2.7.2.3. OPRF in the UC Model

We recall the security notion defined in [JKX18]. The security is defined in the UC-framework, see Section 2.4. We describe the ideal functionality $\mathcal{F}^*_{\mathrm{OPRF}}$ in Figure 2.15. We will write $\mathcal{F}^*_{\mathrm{OPRF}}$ to distinguish this functionality, from the slightly simplified functionality $\mathcal{F}_{\mathrm{OPRF}}$, we are going to introduce in Section 3.2.

The intuition of the functionality is that users interact with servers in several sessions. A session is indexed by an id *sid* an belongs to one user and one server. An honest server uses the same key for the whole session *sid*. The user can request an output of the PRF by interacting with the server in a subsession, identified by *ssid*. The user starts the request of an output $\mathsf{F}_k(x)$ by sending $(\textsc{Eval}, sid, ssid, \mathsf{S}', x)$ to $\mathcal{F}^*_{\mathrm{OPRF}}$. S' denotes the server from which the user wants to get the output. In other words, the user specifies the function $f_k(\cdot)$ from which the output should be taken, only that the user doesn't know the value k but rather specifies the server that holds k. As we assume that a server only holds one k for every session *sid*, the ideal functionality denotes its internal function as $\mathsf{F}_{sid,\mathsf{S}}(\cdot)$. The function is seen as an initially empty table and gets lazily filled with randomly drawn values.

A server can consent to the interaction with the user by sending $(\textsc{SndrComplete}, sid, ssid')$ to $\mathcal{F}^*_{\mathrm{OPRF}}$. Finally, the adversary can send $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, i)$ to $\mathcal{F}^*_{\mathrm{OPRF}}$ to indicate that the user U can receive the requested output. However, $\mathcal{F}^*_{\mathrm{OPRF}}$ gives the adversary the means to tamper with the output by specifying an identity i. This i indicates from which function $\mathsf{F}_{sid,i}(\cdot)$ the output should actually be chosen. If the adversary sends $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S}')$, where S' is the server from the user's $(\textsc{Eval}, sid, ssid, \mathsf{S}', x)$ message, the interaction yields exactly the output that the user requested. But if $i \neq \mathsf{S}'$, the request is "detoured" and the user receives an output from a different table, namely $\mathsf{F}_{sid,i}(\cdot)$. The identity i does not need to correspond to an existing protocol party, but can by any identity label, e.g. any bit string of a predefined length.

The above might give the impression that $\mathcal{F}^*_{\mathrm{OPRF}}$ undermines the security of OPRF protocols realizing $\mathcal{F}^*_{\mathrm{OPRF}}$. If the adversary can arbitrarily detour queries, it could e.g. answer all queries with just one function $\mathsf{F}_{sid,\mathsf{S}}(\cdot)$. This problem is solved via the ticket counter $\mathrm{tx}(\cdot)$. With this counter, $\mathcal{F}_{\mathrm{OPRF}}$ keeps track of the number of OPRF outputs that a server generates and the number of OPRF output from that server that is used as output. Everytime the server consents to giving OPRF output by sending $(\textsc{SndrComplete}, sid, ssid, \mathsf{S})$, the counter $\mathrm{tx}(\mathsf{S})$ is incremented. If an output from S is delivered to a user by a $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message, the counter $\mathrm{tx}(\mathsf{S})$ is decremented. If the counter is zero but an output is request by a $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message, $\mathcal{F}^*_{\mathrm{OPRF}}$ ignores this message.

$\mathcal{F}^*_{\mathrm{OPRF}}$ also allows offline evaluation of functions, by sending $(\textsc{OfflineEval}, sid, i, x)$ to $\mathcal{F}^*_{\mathrm{OPRF}}$. This is possible in four cases:

1. If the server i is corrupted. This models the fact that the adversary learns the PRF key k by corrupting a server. When the adversary knows k, it can evaluate $\mathsf{F}_k(\cdot)$ at arbitrary points.

2. If the server itself wants to evaluate the function, it can do that, as it knows its own key.

3. A real-world adversary can just makeup random output values. This is reflected by the fact that the adversary can send offline evaluation requests for identities i that are not an existing party. For the “virtual corrupt identities”, the adversary can arbitrarily often query output values.

4. If the server is compromised, we are in a similar situation as in the case of corruption.

Note, that $\mathcal{F}^*_{\mathrm{OPRF}}$ models several users and several servers, interacting with each other. This is rather unusual for a UC-functionality as it makes the security analysis more complicated. However, modeling the functionality with only one user and one sender has a drawback. The 2HashDH by [Jar+16; JKK14] relies on two hash functions. More formally speaking, 2HashDH UC-realizes $\mathcal{F}^*_{\mathrm{OPRF}}$ in the *RO*-hybrid model. Now, if different users would want to query pseudo-random values from the same server and thus, the same function $\mathsf{F}_k(\cdot)$, it would not be possible, as the random oracles H_1^{sid}, H_2^{sid} are different for every session and thus the PRF $\mathsf{F}_k(x) = H_2^{sid}(x, (H_1^{sid}(x))^k)$, too.

Functionality $\mathcal{F}^*_{\text{OPRF}}$

Public Parameters: PRF output-length l, polynomial in the security parameter λ. *Conventions:* For every i, x, value $\mathsf{F}_{sid,i}(x)$ is initially undefined, and if undefined value $\mathsf{F}_{sid,i}(x)$ is referenced then $\mathcal{F}^*_{\text{OPRF}}$ assigns $\mathsf{F}_{sid,i}(x) \xleftarrow{\$} \{0,1\}^l$.

Initialization:
On $(\textsc{Init}, sid)$ from S, if this is the first Init message for sid, set $\text{tx} = 0$ and send $(\textsc{Init}, sid, \text{S})$ to $\mathcal{A}$. From now on, use tag "S" to denote the unique entity which sent the Init message for session id sid. Ignore all subsequent Init messages for sid.

Server Compromise:
On $(\textsc{Compromise}, sid, \text{S})$ from $\mathcal{A}$, mark S as Compromised. If S is corrupted, it is marked as Compromised from the beginning. *Note: Message $(\textsc{Compromise}, sid, \text{S})$ requires permission from the environment.*

Offline Evaluation:
On $(\textsc{OfflineEval}, sid, i, x)$ from $\mathsf{P} \in \{\text{S}, \mathcal{A}\}$, send $(\textsc{OfflineEval}, sid, \mathsf{F}_{sid,i}(x))$ to P if any of the following hold: (i) S is corrupted, (ii) $\mathsf{P} = \text{S}$ and $i = \text{S}$, (iii) $\mathsf{P} = \mathcal{A}$ and $i \neq \text{S}$, (iv) $\mathsf{P} = \mathcal{A}$ and S is as marked Compromised.

Evaluation:

- On $(\textsc{Eval}, sid, ssid, \text{S}', x)$ from $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$, send $(\textsc{Eval}, sid, ssid, \mathsf{P}, \text{S}')$ to $\mathcal{A}$. On prfx from $\mathcal{A}$, ignore this message if prfx was used before. Else record $\langle ssid, \mathsf{P}, x, \text{prfx}\rangle$ and send $(\textsc{Prefix}, sid, ssid, \text{prfx})$ to P.
- On $(\textsc{SndrComplete}, sid, ssid)$ from S, send $(\textsc{SndrComplete}, sid, ssid, \text{S})$ to $\mathcal{A}$. On prfx$'$ from $\mathcal{A}$, send $(\textsc{Prefix}, sid, ssid, \text{prfx}')$ to S. If there is a record $\langle ssid, \mathsf{P}, x, \text{prfx}\rangle$ for $\mathsf{P} \neq \mathcal{A}$ and $\text{prfx} \neq \text{prfx}'$, change it to $\langle ssid, \mathsf{P}, x, \text{OK}\rangle$. Else set tx + +.
- On $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ from $\mathcal{A}$, ignore this message if there is no record $\langle ssid, \mathsf{P}, x, \text{prfx}\rangle$ or if ($i = \text{S}$, $\text{tx} = 0$ and $\text{prfx} \neq \text{OK}$). Else send $(\textsc{EvalOut}, sid, ssid, \mathsf{F}_{sid,i}(x))$ to P and if ($i = \text{S}$ and $\text{prfx} \neq \text{OK}$) then set tx − −.

Figure 2.15.: The Ideal Functionality $\mathcal{F}^*_{\text{OPRF}}$ From [JKX18].

3. Construction

3.1. Adversarial Model

For the sake of clarity, we formulate the assumptions about our adversaries:

We will implement an OPRF with garbled circuits. As "textbook versions" of garbled circuits offer only security against passive, i.e., semi-honest adversaries, we will restrict our construction to these adversaries. This means, the adversary follows the protocol honestly but tries to learn additional information from its view on the protocol execution. Further, we will restrict ourselves to a model of static corruption. This means the adversary can only at the start of the protocol choose to gain control over certain parties. If a party is corrupted, we assume that the adversary learns the party's input, the content of the party's random tape, and all messages received by the party. The adversary can send messages in the name of a corrupted party as long as the messages adhere to the protocol.

3.2. Security Notion

We will not use exactly the same formulation of the ideal OPRF functionality $\mathcal{F}^*_{\mathrm{OPRF}}$, defined in Section 2.7. We'll use a slightly simplified version, described in Figure 3.1. Note, that $\mathcal{F}_{\mathrm{OPRF}}$ does not capture adaptive compromise, as we only assume static corruption. For the sake of simplicity, we also omit the prefixes used in $\mathcal{F}^*_{\mathrm{OPRF}}$.

3.3. The main construction

Let $m, n \in \Omega(\lambda)$ and $F : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ be a PRF, with the additionally property that for every $k \in \{0,1\}^m$ it holds that $F_k(\cdot) : \{0,1\}^n \to \{0,1\}^n$ is a permutation. In our real-world implementation, described in Chapter 5, we instantiate this function with AES. We will garble the circuit C that describes F to construct our OPRF.

The user runs with its password $pw \in \{0,1\}^*$ as input. The password is hashed to an n bit value, so we can use it as input to C. Our construction involves two hash functions $H_1 : \{0,1\}^* \to \{0,1\}^n$ and $H_2 : \{0,1\}^* \times \{0,1\}^m \to \{0,1\}^l$, where $l \in \Omega(\lambda)$. We will model these hash functions as random oracles. The server takes no input. Initially, for each session, it chooses a key $k \in \{0,1\}^m$ uniformly at random. The PRF, that is computed by the OPRF protocol is

$$\mathsf{F}_k(pw) := H_2(pw, C_k(H_1(pw))).$$

In our description of the protocol, the server garbles the circuit and the user evaluates the circuit. The user starts an execution of the protocol by hashing its input pw. The obtained value $x = H_1(pw)$ will be used as the user's input to the circuit. The user then requests

Functionality $\mathcal{F}_{\text{OPRF}}$

Initialization:
For each value i and each session sid, an empty table $T_{sid}(i,\cdot)$ is initially undefined. Whenever $T_{sid}(i,x)$ is referenced below while it is undefined, draw $T_{sid}(i,x) \overset{\$}{\leftarrow} \{0,1\}^l$.

On $(\textsc{Init}, sid)$ from S, if this is the first $\textsc{Init}$ message for sid, set $\text{tx}(\mathsf{S}) = 0$ and send $(\textsc{Init}, sid, \mathsf{S})$ to $\mathcal{A}$. From now on, use "S" to denote the unique entity which sent the $\textsc{Init}$ message for sid. Ignore all subsequent $\textsc{Init}$ messages for sid.

Offline Evaluation:
On $(\textsc{OfflineEval}, sid, i, x)$ from $\mathsf{P} \in \{\mathsf{S}, \mathcal{A}\}$, send $(\textsc{OfflineEval}, sid, T_{sid}(i,x))$ to P if any of the following hold: (i) S is corrupted and $i = \mathsf{S}$, (ii) $\mathsf{P} = \mathsf{S}$ and $i = \mathsf{S}$, (iii) $\mathsf{P} = \mathcal{A}$ and $i \neq \mathsf{S}$.

Online Evaluation:

- On $(\textsc{Eval}, sid, ssid, \mathsf{S}, pw)$ from $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$, record $\langle ssid, \mathsf{S}, \mathsf{P}, pw\rangle$ and send $(\textsc{Eval}, sid, ssid, \mathsf{P}, \mathsf{S})$ to $\mathcal{A}$.
- On $(\textsc{SndrComplete}, sid, ssid)$ from S, increment $\text{tx}(\mathsf{S})$ or set to 1 if previously undefined, send $(\textsc{SndrComplete}, sid, ssid, \mathsf{S})$ to $\mathcal{A}$.
- On $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ from $\mathcal{A}$, retrieve $\langle ssid, \mathsf{S}, \mathsf{P}, pw\rangle$, where $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$. Ignore this message if at least one of the following holds:
  - There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, pw\rangle$.
  - $i = \mathsf{S}$ but $\text{tx}(\mathsf{S}) = 0$.
  - S is honest but $i \neq \mathsf{S}$.

  Send $(\textsc{EvalOut}, sid, T_{sid}(i, pw))$ to P. If $i = \mathsf{S}$ set $\text{tx}(i) - -$.

Figure 3.1.: The Ideal Functionality $\mathcal{F}_{\text{OPRF}}$ Inspired by [JKX18].

a garbled circuit from the server by sending $(\textsc{Garble}, sid, ssid)$ to the server. The server will proceed by calculating the garbled circuit, using half-gates, as described in Figure 2.12. In particular, it encodes its own key as input for the circuit. It sends the garbled circuit, the input labels of the key, and the decoding information to the user. The user and the server perform n parallel 1-out-of-2-OTs in order to equip the user with the wire labels for its desired input $x = H_1(pw)$. Next, the user can evaluate the garbled circuit on the encoded inputs X and K and receives an output label Y. This label can be decoded to obtain the output value of the circuit y. Finally, the user hashes its input and the output of the circuit again to obtain the output $\rho = H_2(pw, y)$. We describe the OPRF more precisely in Figure 3.2. We denote by *mid* the session id of each $\mathcal{F}_{\text{AUTH}}$ session, i.e., each sent message. We assume that *mid* contains the session id *sid* and the subsession id *ssid* as a substring. When we talk about the labels generated by Gb, we will write $X[0]$ (or $X[1]$, rsp.) to denote that the label is an encoding of 0 (or 1, rsp.). When $b \in \{0,1\}^n$, we will also write $X[b]$ to denote the string of labels $X[b_1] \| \ldots \| X[b_n]$.

3.4. Some Remarks on the Construction

In the following, we give some remarks on the construction and explain decisions on the protocol design.

Who garbles? We believe that the above-described approach could easily be adapted to feature switched roles of garbler and evaluator. More precisely, we believe that it's also possible to construct a similar OPRF protocol where the user garbles the circuit and the server evaluates the circuit. However, we decided to let the server garble the circuit because our construction only has passive security. If the protocol would be implemented in a real-world scenario, it is a more realistic assumption that a server behaves in an honest-but-curious way than to assume that a user behaves that way. A server might be maintained by a company that would fear economic damage if malicious behavior of their servers is uncovered, while arbitrary users on the internet are likely to behave maliciously. Nonetheless, we would always recommend using protocols that feature security against active adversaries for real-world scenarios. If it would be possible to achieve an actively secure OPRF protocol from garbled circuits, it might even be beneficial to switch roles. If the user has to "invest" computation time on the creation of a garbled circuit, it decreases the thread of Denial of Service (DOS) attacks on the server.

On the Need for the Second Hash Function One might ask why we need a second hash function H_2 in the definition of our pseudo-random function $\mathsf{F}_k(x) = H_2(C_k(H_1(x)))$. On the first glance it even seems to weaken our results, as the construction in Figure 3.2 is only a *weak* OPRF, see Section 2.7. One could conclude that if the user would not have to hash the output of the garbled circuit, we would achieve a *strong* OPRF, as the user does not learn anything more than the PRF output, instead of learning the H_2 pre-image of the actual output. The pseudo-randomness would follow from the fact that C is a PRF. That would lead to the OPRF described by Pinkas et al. [Pin+09]. The problem with this lies in the definition of the ideal functionality, see Figure 2.15, and the strong notion of universal

```
S on ($\textsc{Init}, sid$) from $\mathcal{E}$

If this is the first ($\textsc{Init}, sid$) message from $\mathcal{E}$
$k \xleftarrow{\$} \{0,1\}^m$, record $\langle k, sid \rangle$

U on ($\textsc{Eval}, sid, ssid, \mathrm{S}, pw$) from $\mathcal{E}$

$x \xleftarrow{\$} H_1(pw)$
send ($\textsc{Send}, mid, \mathrm{S}, (\textsc{Garble}, sid, ssid)$) to $\mathcal{F}_{\mathrm{AUTH}}$

S on ($\textsc{SndrComplete}, sid, ssid$) from $\mathcal{E}$

if already received ($\textsc{Garble}, sid, ssid$):
  goto GarbleCircuit
else
  ignore this message

S on ($\textsc{Sent}, mid, \mathrm{U}, \mathrm{S}, (\textsc{Garble}, sid, ssid)$) from $\mathcal{F}_{\mathrm{AUTH}}$

if already received ($\textsc{SndrComplete}, sid, ssid$):
  GarbleCircuit:
  if $\nexists \langle k, sid \rangle$:
    ignore this message
  $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$
  $(X[0^n] \,\|\, K) := \mathsf{En}(e, 0^n \,\|\, k)$
  $(X[1^n] \,\|\, K) := \mathsf{En}(e, 1^n \,\|\, k)$
  send ($\textsc{Send}, mid', \mathrm{U}, (sid, ssid, (F, K, d))$) to $\mathcal{F}_{\mathrm{AUTH}}$
  for $i \in \{1, \ldots, n\}$ :
    send ($\text{OT-}\textsc{Send}, (ssid, i), (X_i[0], X_i[1])$) to $\mathcal{F}_{\mathrm{OT}}$
else
  ignore this message

U on ($\textsc{Sent}, mid, \mathrm{S}, \mathrm{U}, (sid, ssid, (F, K, d))$) from $\mathcal{F}_{\mathrm{AUTH}}$

if already received ($\textsc{Eval}, sid, ssid, \mathrm{S}, pw$):
  wait for ($\text{OT-}\textsc{Sent}, (ssid, 1)$), ..., ($\text{OT-}\textsc{Sent}, (ssid, n)$) from $\mathcal{F}_{\mathrm{OT}}$
  for $i \in \{1, \ldots, n\}$ :
    send ($\text{OT-}\textsc{Receive}, (ssid, i), x_i$) to $\mathcal{F}_{\mathrm{OT}}$
else
  ignore this message

U on ($\text{OT-}\textsc{Received}, (ssid, 1), X_1$), ..., ($\text{OT-}\textsc{Received}, (ssid, n), X_n$) from $\mathcal{F}_{\mathrm{OT}}$

if already received ($sid, ssid, (F, K, d)$):
  $Y := \mathsf{Ev}(F, X \,\|\, K)$
  $y := \mathsf{De}(d, Y)$
  $\rho \xleftarrow{\$} H_2(pw, y)$
  output ($\textsc{EvalOut}, sid, ssid, \rho$) to $\mathcal{E}$
else
  ignore this message
```

Figure 3.2.: Our GC-OPRF Construction in the $\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{RO}}, \mathcal{F}_{\mathrm{AUTH}}$-Hybrid Model.

composability. We argue in an informal way, why the Pinkas et al. [Pin+09] OPRF protocol does not UC-realize the ideal functionality $\mathcal{F}_{\mathrm{OPRF}}$ from Figure 2.15.

For a newly queried value, $\mathcal{F}_{\mathrm{OPRF}}$ draws a fresh output value uniformly at random. This means that the OPRF output of the real protocol must be indistinguishable from a truly random function for *every environment*. Indeed, we assume that the garbled circuit is a PRF so the output of the circuit should be indistinguishable from random values. But this does not hold if the PRF key is known. Let's imagine an environment that corrupted a server. That means the environment knows the key k of that server. Next, the environment could query a value $H_1(x) = h$ and as the description of the garbled circuit C is public, the environment can calculate $y = C_k(h)$. Now the environment can start a protocol execution between an honest user with input x and the corrupted server with key k. In the ideal world, the functionality $\mathcal{F}_{\mathrm{OPRF}}$ will draw a uniformly random value as output for the user. However, that output will be independent of the output y that the environment calculated beforehand, making it easy to distinguish the real and the ideal world. So a simulator needs some way to manipulate the output accordingly. One might think of programming the RO for H_1. However, this does not seem to suffice, as $H_1(x)$ can only be programmed once, while an environment could easily repeat the above experiment for several corrupted servers with different keys but with the same input x. The solution we use is to introduce the second hash function H_2. This hash function allows the simulator to program the output of the circuit to fit the outputs generated by $\mathcal{F}_{\mathrm{OPRF}}$.

On the Need for Authenticated Channels In the proof of security in Section 3.5, we assume authenticated channels. This is necessary, as otherwise, we could not rely on the semi-honest nature of messages sent to the simulator. By assuming that all parties behave honest-but-curious, we do explicitly not mean the adversary. In this model, the adversary could still send e.g. malformed circuits in lieu of the honestly generated circuit from the server. To really get to a setting where the simulator can be sure of all the messages being benign, we must make this additional assumption.

One could argue that the assumption of authenticated channels renders our construction impractical for many settings. For instance, if the OPRF is used for password-based authentication, as we discussed in Chapter 1, one might not necessarily expect to already have an authenticated channel. But in fact, authenticated channels are already established in many practical scenarios! Typically, a user would connect to a server over a TLS channel, and thus, at least the server is authenticated via digital certificates. A user can also authenticate itself to the server with a certificate. We even expect the security of our construction holds if only the server authenticates itself. This does guarantee that the garbled circuit was actually generated by the party with which the user intends to communicate.

If the server implements an OPRF protocol for its own password-based authentication mechanism, our protocol is still useful. Imagine for example a typical internet forum. Users will connect to the website via Hypertext Transfer Protocol Secure (HTTPS) but then use a username and password to log in to their forum account. The big security benefit is that the user's password is protected even if the server is compromised should be motivation enough to use a protocol like OPAQUE [JKX18]. Clearly, a protocol that assumes authenticated channels cannot be used to establish a TLS session. But TLS relies mostly on a PKI and certificates instead of password-based authentication.

3.5. Proving Security

In order to prove that GC-OPRF actually UC-emulates $\mathcal{F}_{\mathrm{OPRF}}$ in the $\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{RO}}, \mathcal{F}_{\mathrm{AUTH}}$-hybrid model, we have to compare the views of two protocol executions. More precisely, for every adversary $\mathcal{A}$ we must specify a simulator Sim such that for every environment $\mathcal{E}$ we have:

$$\mathrm{EXEC}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{OPRF}}},\mathsf{Sim},\mathcal{E}} \overset{c}{\approx} \mathrm{EXEC}_{\mathsf{GC\text{-}OPRF},\mathcal{A},\mathcal{E}},$$

where $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{OPRF}}}$ denotes the ideal protocol execution.

As discussed in Section 2.4, we will only consider a Dummy-Adversary $\mathcal{A}$. We construct the simulator as in Figures 3.5 to 3.8. For the sake of readability, we split the description of Sim into four figures. We denote parties with a hat, e.g. $\hat{\mathsf{P}}$, if it is clear from the context that they are corrupted. We write $\exists\langle r\rangle$ as shorthand for "Sim checks if a record $\langle r\rangle$ exists".

Some Intuition on the Simulator Before we give a formal proof, we like to give some intuition on the simulator in Figures 3.5 to 3.8. First, note that in the formulation of the UC security experiment in Section 2.4, the simulator Sim replaces the adversary $\mathcal{A}$. That means all messages the environment sends to $\mathcal{A}$ will be received by Sim. We also assume that the real-world adversary $\mathcal{A}$ is a dummy adversary, as elaborated in Section 2.4. Nonetheless, we write in Figures 3.5 to 3.8 as if there was a party "$\mathcal{A}$". By this we mean the messages Sim receives from $\mathcal{E}$ addressed to $\mathcal{A}$ or messages that Sim sends to $\mathcal{E}$ acting as $\mathcal{A}$.

As always in the UC model, the simulator answers all queries addressed to ideal functionalities that were present in the real world. As we are working in the $\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{RO}}, \mathcal{F}_{\mathrm{AUTH}}$-hybrid model, Sim has to simulate $\mathcal{F}_{\mathrm{OT}}$ and $\mathcal{F}_{\mathrm{AUTH}}$. Remember that a random oracle is strictly speaking an ideal functionality, too. We just do not notate it like that for the sake of convenience for the reader. Thus, Sim must also answer queries to the random oracles H_1, and H_2. In the ideal world of the UC security experiment all honest parties just forward the input they receive from the environment $\mathcal{E}$ to the ideal functionality. If they receive output from the ideal functionality, they forward this output to $\mathcal{E}$. However, the adversary can send messages on behalf of corrupted parties, meaning the adversary gets instructed to do so by the environment.

From a high viewpoint, the simulator can be summarized as follows: For honest servers, the simulator chooses internally a PRF key k and follows the protocol exactly as a real server would do with key k. For an honest user, the simulator requests a garbled circuit from the server and simulates the request of input labels via OT. Note here, that Sim does not know the input of the user. It can simulate the messages anyway as Sim does also act as $\mathcal{F}_{\mathrm{OT}}$. Then Sim receives a garbled circuit and input labels but for every input $x_i \in \{0,1\}$ bit, Sim receives both labels $X_i[0]$ and $X_i[1]$, again because Sim simulates $\mathcal{F}_{\mathrm{OT}}$. Sim requests an output for the user from $\mathcal{F}_{\mathrm{OPRF}}$. Now, $\mathcal{F}_{\mathrm{OPRF}}$ makes the user output some uniformly random value, and Sim programs $H_2(p, y)$ accordingly. As we will see, correct programming is non-trivial.

H_2 must be programmed because the output of a user in the real world is always the output of $H_2(p, y)$ for some values p and y. However, in the ideal world, the output for honest users is generated by the ideal functionality $\mathcal{F}_{\mathrm{OPRF}}$. Hence, the simulator must ensure that the output generated by $\mathcal{F}_{\mathrm{OPRF}}$ and $H_2(p, y)$ coincide for values of p and y that can occur in an execution of the protocol. Sim can query output from $\mathcal{F}_{\mathrm{OPRF}}$ but this has to be done carefully as $\mathcal{F}_{\mathrm{OPRF}}$ maintains a ticket counter that ensures that not more PRF values can be received than server executions were performed. Especially, Sim must somehow identify a server holding the key k that mapped p to $y = C_k(H_1(p))$. We argue in the following, why Sim has to do this.

Let's assume Sim would always choose the same server identity i to receive its output from. Clearly, $\mathcal{F}_{\mathrm{OPRF}}$ would ignore the requests of Sim as soon as $\mathcal{E}$ would query $(\textsc{Eval}, sid, ssid, \mathsf{U}, \mathsf{S}', x)$ from a server $\mathsf{S}' \neq i$ for which $\mathcal{E}$ also sent $(\textsc{SndrComplete}, sid, ssid, \mathsf{S}')$. This is, because $(\textsc{SndrComplete}, sid, ssid, \mathsf{S}')$ increments the ticket counter $\mathrm{tx}(\mathsf{S}')$ of $\mathcal{F}_{\mathrm{OPRF}}$ by one. In contrast, if Sim queries output from $\mathcal{F}_{\mathrm{OPRF}}$ by sending $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, i)$ that decrements the ticket counter $\mathrm{tx}(i)$ of i and not S'. Remember, that $\mathcal{F}_{\mathrm{OPRF}}$ ignores a $\textsc{RcvCmplt}$ request when a ticket counter $\mathrm{tx}(i)$ would be decremented below 0 for some i.

One might be tempted to try the other extreme instead. What happens if the simulator uses a completely new identity i for each and every new query? We will call this i a "corrupt virtual identity". By that we mean the following: Sim can query PRF output for a server identity i from $\mathcal{F}_{\mathrm{OPRF}}$ if this i is no identity of an actual server of the session. See *OfflineEval* point (iii) in Figure 3.1. These corrupt virtual identities do not have a ticket counter. By using sending $(\textsc{OfflineEval}, sid, i, x)$ to $\mathcal{F}_{\mathrm{OPRF}}$, the simulator receives the entry $T_{sid}(i, x)$ from $\mathcal{F}_{\mathrm{OPRF}}$'s table. This identity i must not correspond to an actually existent server in that session sid. Why can't Sim create a new such corrupt virtual identity for every H_2 query it receives? Consider the following counter example:

$\mathcal{E}$ chooses a corrupted server S^* with key $k \in \{0,1\}^m$. The circuit C is publicly known, so $\mathcal{E}$ can precompute $C_k(x_0) = y_0$ and $C_k(x_1) = y_1$ where $x_0 = H_1(m_0)$ and $x_1 = H_1(m_1)$ for two messages $m_0, m_1 \in \{0,1\}^*$. Now $\mathcal{E}$ lets $\mathcal{A}$ query $H_2(m_0, y_0)$ from Sim. Note that there was no protocol execution so far and hence, Sim has not received nor calculated a garbled circuit (F, K, d). As we assumed in the beginning, Sim now creates a new "virtual corrupt identity". In other words, Sim creates a new identity i for which no prior queries to $\mathcal{F}_{\mathrm{OPRF}}$ exist. Now, Sim sends $(\textsc{OfflineEval}, sid, i, m_0)$ to $\mathcal{F}_{\mathrm{OPRF}}$. As i is no identity of an actual server, $\mathcal{F}_{\mathrm{OPRF}}$ will answer with $(\textsc{OfflineEval}, sid, \rho_0 := T_{sid}(i, m_0))$. Like we assumed in the begining, Sim programs $H_2(m_0, y_0) := \rho_0$. Now $\mathcal{E}$ repeats this for $H_2(m_1, y_1)$. Sim will query $(\textsc{OfflineEval}, sid, i', m_1)$ to $\mathcal{F}_{\mathrm{OPRF}}$ for $i' \neq i$ and will receive $(\textsc{OfflineEval}, sid, \rho_1 := T_{sid}(i', m_1))$ and set $H_2(m_1, y_1) := \rho_1$. Next, suppose $\mathcal{E}$ starts a protocol execution between the server S^* and an honest user U with input m_b, where $b \in \{0,1\}$ is a secret bit, only $\mathcal{E}$ knows. As S^* is corrupted, $\mathcal{E}$ will send a garbled circuit (F, K, d) and input labels $X_1[0], X_1[1], \ldots, X_n[0], X_n[1]$ to Sim. Sim has no information about x_b, as the honest U's input is "protected" by the security of the OT protocol, see Figure 2.7, and the privacy of the garbled circuit, see Definition 11. However, Sim must produce an output for U by sending some message $(\textsc{RcvCmplt}, \ldots)$ to $\mathcal{F}_{\mathrm{OPRF}}$, because an honest user in the real world would also output something after receiving the garbled circuit and the labels. Sim could create an new "virtual corrupt identity" i''. However, $\mathcal{F}_{\mathrm{OPRF}}$'s answer $T_{sid}(i'', m_b)$ would be different from ρ_b with high probability, as $T_{sid}(i'', m_b)$ is a uniformly random value. Alternatively, Sim could go through all prior $H_2(\cdot,\cdot)$ queries and check for each query (α,β) if $\beta = \mathsf{De}(d, \mathsf{Ev}(F, X[H_1(\alpha)] \parallel K))$. Intuitively, that indicates that the key K "maps" $H_1(\alpha)$ to β, i.e., $C_k(H_1(\alpha)) = \beta$ if K encodes k. Sim would find that it already received a query $H_2(m_b, y_b)$, such that $y_b = \mathsf{De}(d, \mathsf{Ev}(F, X[H_1(m_b)] \parallel K))$. The problem is, that Sim would also find the second query $H_2(m_{1-b}, y_{1-b})$ for which it holds that $y_{1-b} = \mathsf{De}(d, \mathsf{Ev}(F, X[H_1(m_{1-b})] \parallel K))$. In that case, Sim must guess b. Because if $b = 0$, the result must be queried as $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, i)$ and if $b = 1$, the result must be queried as $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, i')$. If Sim guesses wrong, $\mathcal{E}$ can distinguish this protocol execution from an real execution. Because if we denote by $\langle \mathsf{U}(m_b), \mathsf{S}^* \rangle_{\mathsf{U}}$ the output of U on input m_b when interacting with server S^*, $\mathcal{E}$ sees that $H_2(m_b, y_b) \neq \langle \mathsf{U}(m_b), \mathsf{S}^* \rangle_{\mathsf{U}}$. As Sim has no information about b this happens with probability $1/2$. This example makes clear, why care must be taken when programming the random oracle H_2.

Our strategy for programming $H_2(p, y)$ is the following: If Sim receives a query, it looks up the corresponding H_1 query $H_1(p) = h$. If no such query exists, Sim can safely set $H_2(p, y)$ to a uniformly random value. If such a query exists, Sim knows the input value h for the circuit. Now, it checks if there either was an honest server or a corrupted server, such that $y = C_k(h)$ holds for the key k of one of the servers. For an honest server, Sim requests the output value from $\mathcal{F}_{\mathrm{OPRF}}$ by sending a $\textsc{RcvCmplt}$ message and for a corrupted server, Sim requests the output value from $\mathcal{F}_{\mathrm{OPRF}}$ with an $\textsc{OfflineEval}$ message.

Proof Strategy In the ideal world the environment can control the execution by sending messages to the parties in the following ways:

- Honest user U: The environment $\mathcal{E}$ sends $(\textsc{Eval}, sid, ssid, \mathsf{S}, pw)$ messages to U. User U transmits this message to $\mathcal{F}_{\mathrm{OPRF}}$ and outputs $(\textsc{EvalOut}, sid, ssid, \rho)$ to $\mathcal{E}$.
- Honest server S:
  - $\mathcal{E}$ sends $(\textsc{Init}, sid)$ to S. Server S transmits this message to $\mathcal{F}_{\mathrm{OPRF}}$ who sends $(\textsc{Init}, sid, \mathsf{S})$ to $\mathcal{A}$. Im Realen wird die Nachricht einfach ignoriert, oder?
  - $\mathcal{E}$ sends $(\textsc{SndrComplete}, sid, ssid)$ to S. Server S forwards this message to $\mathcal{F}_{\mathrm{OPRF}}$. The functionality $\mathcal{F}_{\mathrm{OPRF}}$ forwards this message to $\mathcal{A}$.
- Dummy adversary $\mathcal{A}$:
  - The environment can send $(\textsc{Send}, mid, \mathsf{S}, (\textsc{Garble}, sid, ssid))$, and $(\text{OT-}\textsc{Receive}, (ssid, i), x_i)$ to $\mathcal{A}$. The adversary $\mathcal{A}$ acts as corrupted user $\hat{\mathsf{U}}$ and forwards these messages to Sim. $\mathcal{A}$ sends all responses it receives to $\mathcal{E}$.
  - The environment can send $(\textsc{Send}, mid, \mathsf{U}, (sid, ssid, (F, K, d))$, and $(\text{OT-}\textsc{Send}, (ssid, i), (X_i[0], X_i[1])$ to $\mathcal{A}$. The adversary $\mathcal{A}$ acts as corrupted server $\hat{\mathsf{S}}$ and $\mathcal{A}$ forwards these messages to Sim. Again, $\mathcal{A}$ sends all responses it receives to $\mathcal{E}$.
  - The environment can send $(\text{OT-}\textsc{Sent}, (ssid, i))$, $(\text{OT-}\textsc{Received}, (ssid, i))$, and $(\textsc{ok}, mid)$ to $\mathcal{A}$. The adversary $\mathcal{A}$ will send these messages to Sim, acting as adversary.

The view of the environment $\mathcal{E}$ is comprised of all messages that $\mathcal{E}$ receives as a reaction to one of the messages above. The following messages form the view of the environment:

- ($\textsc{EvalOut}, sid, ssid, \rho$) from U as response to an ($\textsc{Eval}, sid, ssid, \mathsf{S}, pw$) message.
- ($\textsc{Sent}, mid, \mathsf{U}, \mathsf{S}, (\textsc{Garble}, sid, ssid)$) from $\mathcal{A}$ when $\mathcal{A}$ acts as server and receives this message, formatted as being sent from a user via $\mathcal{F}_{\mathrm{AUTH}}$.
- ($\textsc{Sent}, mid, \mathsf{S}, \mathsf{U}, (sid, ssid, (F, K, d)$)) from $\mathcal{A}$ when $\mathcal{A}$ acts as user and receives this message, formatted as being sent from a server via $\mathcal{F}_{\mathrm{AUTH}}$.
- ($\text{OT-}\textsc{Send}, (ssid, i)$) from $\mathcal{A}$ when a server sends two messages to Sim, who acts as $\mathcal{F}_{\mathrm{OT}}$.
- ($\text{OT-}\textsc{Receive}, (ssid, i)$) from $\mathcal{A}$ when a user sends a choice bit to Sim, who acts as $\mathcal{F}_{\mathrm{OT}}$.
- ($\text{OT-}\textsc{Sent}, sid$) from $\mathcal{A}$ when $\mathcal{A}$ acts as server and sent ($\text{OT-}\textsc{Send}, sid, (X_0, X_1)$) to Sim before. Sim acts as $\mathcal{F}_{\mathrm{OT}}$.
- ($\text{OT-}\textsc{Received}, sid, x_b$) from $\mathcal{A}$ when $\mathcal{A}$ acts as user and sent ($\text{OT-}\textsc{Receive}, sid, b$) to Sim before. Sim acts as $\mathcal{F}_{\mathrm{OT}}$.
- Responses to $H_1(\cdot)$ and $H_2(\cdot, \cdot)$ queries from $\mathcal{A}$.

Our goal in the following proof is to argue, why the above-described view of the environment in the ideal world is computationally indistinguishable from the view of the environment in the real world. We construct a simulator such that each message in the real world, has a directly corresponding message in the ideal world. Loosely speaking, the simulator creates messages that “look the same” as in the real world. For instance, Sim sends a message ($\textsc{Garble}, sid, ssid$) that is formatted exactly like a ($\textsc{Garble}, sid, ssid$) message sent by the user in the real world. Further, Sim ensures that the messages are sent in the same circumstances, i.e., at the same time. For example, Sim will send ($\textsc{Garble}, sid, ssid$) when an honest user is invoked by $\mathcal{E}$ with ($\textsc{Eval}, sid, ssid, \mathsf{S}, pw$), as this is how the real-world user would react. The main idea is that the view in the real world is indistinguishable from the view in the ideal world, if each message in the real world is indistinguishable from its corresponding message in the ideal world.

We cannot analyze the protocol with a single distinction of cases in the style of “(1) both parties are honest, (2) only user is corrupted, (3) only server is corrupted, (4) both parties are corrupted.” This is because the ideal functionality Figure 3.1 – and also Figure 2.15 from [JKX18] – handles multiple users interacting with multiple servers. Therefore, we will only consider one simulator Sim that has to keep records of messages it gets to “dynamically” decide for each message which situation Sim must simulate.

Formal Proof

Theorem 1. *Let the garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ *have* privacy*, as defined in Definition 11.Let* C *denote the boolean circuit of a PRF. Then* GC-OPRF *UC-realizes* $\mathcal{F}_{\mathrm{OPRF}}$ *in the* $\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{AUTH}}, \mathcal{F}_{\mathrm{RO}}$*-hybrid model.*

Proof. As explained above, we will argue for each message that $\mathcal{E}$ receives why it is indistinguishable for $\mathcal{E}$ whether the message comes from a real protocol execution or the ideal execution with the simulator.

Responses to OT messages

- (OT-Send, $(ssid, i)$) from $\mathcal{A}$ when a server sends two messages to Sim:

  This message is exactly formatted as a OT-Send message from the functionality $\mathcal{F}_{\mathrm{OT}}$. Further, Sim behaves exactly like $\mathcal{F}_{\mathrm{OT}}$ in sending those messages. Concretely, on a message (OT-Send, $(ssid, i)$, $(X_i[0], X_i[1])$), Sim stores the labels and informs the adversary that two labels were sent – but not which labels – by sending (OT-Send, $(ssid, i)$) to $\mathcal{A}$. This is exactly the behavior of $\mathcal{F}_{\mathrm{OT}}$, as described in Figure 2.7. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real- or the ideal execution.

- (OT-Receive, $(ssid, i)$) from $\mathcal{A}$ when a user sends a choice bit to Sim:

  A similar comparison as above shows, that Sim behaves exactly like the original $\mathcal{F}_{\mathrm{OT}}$. That means, on a message (OT-Receive, $(ssid, i)$, b), Sim stores the choice bit b and informs the adversary that a choice bit was received, but not which bit. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real or the ideal execution.

- (OT-Sent, sid) from $\mathcal{A}$ when $\mathcal{A}$ acts as server and sent (OT-Send, sid, (X_0, X_1)) to Sim before:

  Again, Sim behaves like $\mathcal{F}_{\mathrm{OT}}$ when creating those messages. Namely, upon receiving a message (OT-Sent, sid) from the adversary, Sim ignores the message if $\langle sid, x_0, x_1\rangle$ or $\langle sid, b\rangle$ is not recorded; Otherwise Sim sends (OT-Sent, sid) to $\hat{\mathrm{S}}$. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real or the ideal execution.

- (OT-Received, sid, x_b) from $\mathcal{A}$ when $\mathcal{A}$ acts as user and sent (OT-Receive, sid, b) to Sim before:

  These are the only messages on which Sim behaves sometimes differently than $\mathcal{F}_{\mathrm{OT}}$. The messages are received by Sim when the adversary "allows the delivery" of OT-messages to the OT-receiver. If Sim recorded a choice bit $x_i \neq \perp$, it means that $\mathcal{A}$ sent (OT-Receive, $(ssid, i)$, x_i) before and Sim answers the query like $\mathcal{F}_{\mathrm{OT}}$ would do. In particular, those queries do not stem from the simulation of a protocol run with an honest user.

  Sim does behave differently than $\mathcal{F}_{\mathrm{OT}}$ in the case when there are n OT-Received messages (OT-Received, $(ssid, 1)$), …, (OT-Received, $(ssid, n)$) with the same value $ssid$, see line 71 in Figure 3.7. The condition means that a complete set of input labels were sent via OT. If further a record $\langle sid, ssid, (F, K, d)\rangle$ exists with the same $ssid$, all information for one OPRF execution was exchanged between server and user. We stress that we assume in this proof that a corrupted server will never send a modified circuit $F' \neq F$, modified decoding information $d' \neq d$, or a modified encoded key $K' \neq K$, where F, K, d are the outputs of Gb and En. Otherwise, the adversary could easily garble a different circuit than F without the user noticing it. This weakness is inherent to "textbook" garbled circuit constructions, see Section 2.6. Further, we know that these labels belong to an interaction with an honest user, as no value $x_i \neq \perp$ was recorded. In the real protocol, a user would evaluate the garbled circuit and output the result as soon as it received all necessary input labels via $\mathcal{F}_{\mathrm{OT}}$. Thus, the simulator must also produce an output for honest users. The simulator retrieves the server identity S connected to *sid*. Sim sends $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S}, \mathsf{S})$ to $\mathcal{F}_{\mathrm{OPRF}}$. The functionality $\mathcal{F}_{\mathrm{OPRF}}$ will ignore this message in any of the three following cases:

1. There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$.
2. $i = \mathsf{S}$ but $\mathrm{tx}(\mathsf{S}) = 0$.
3. S is honest but $i \neq \mathsf{S}$.

The ignore condition of Item 1 cannot occur, as Sim found a record $\langle sid, ssid, (F, K, d)\rangle$. Sim does only create this record, if a corresponding $\langle \textsc{Garble}, sid, ssid\rangle$ record was found. That record in turn is only created when an $(\textsc{Eval}, sid, ssid, \mathsf{U}, \mathsf{S})$ message was received from $\mathcal{F}_{\mathrm{OPRF}}$. We argue in Lemma 1 why the condition of Item 2 occurs at most with negligible probability. The third condition in Item 3 can indeed occur. However, as we assume passive corruption and authenticated channels, a real-world user would also ignore a message $(sid, ssid, (F, K, d))$ that is not from the designated server.

If the RcvCmplt message is not ignored in line 74 in Figure 3.7, the ideal functionality will choose a random values ρ according to its internal random function associated to S as output for U and U will output ρ. We will examine the distribution of ρ in the paragraph "Honest User Output" on Page 45.

Responses to Protocol Messages

- $(\textsc{Sent}, mid, \mathsf{U}, \hat{\mathsf{S}}, (\textsc{Garble}, sid, ssid))$ from $\mathcal{A}$ when $\mathcal{A}$ acts as server and receives this message, formatted as being sent from a user via $\mathcal{F}_{\mathrm{AUTH}}$:

  Sim simulates the behavior of $\mathcal{F}_{\mathrm{AUTH}}$, meaning it informs $\mathcal{A}$ that a message is being sent via $\mathcal{F}_{\mathrm{AUTH}}$ and waits for the delivery until $\mathcal{A}$ sent $(\textsc{ok}, mid)$. The message $(\textsc{Garble}, sid, ssid)$ is sent by Sim as a reaction to an $(\textsc{Eval}, sid, ssid, \mathsf{U}, \hat{\mathsf{S}})$ message from $\mathcal{F}_{\mathrm{OPRF}}$, because a real user would also start a protocol execution by requesting a garbled circuit from $\hat{\mathsf{S}}$. The message itself contains only the session- and subsession id, it is identical in both executions. Thus, we see that $\mathcal{E}$'s view on the $(\textsc{Garble}, sid, ssid)$ message in the real world is is indistinguishable from this message created by Sim.

- $(\textsc{Sent}, mid, \mathsf{S}, \hat{\mathsf{U}}, (sid, ssid, (F, K, d))$ from $\mathcal{A}$ when $\mathcal{A}$ acts as user and receives this message, formatted as being sent from a server via $\mathcal{F}_{\mathrm{AUTH}}$:

  This message is created by Sim, when Sim received a $(\textsc{SndrComplete}, sid, ssid)$ message. If the user of the subsession is corrupted, Sim also expects a $(\textsc{Garble}, sid, ssid)$ message from the user, as in a real execution, the server only starts garbling a circuit when it received both messages. In a subession with an honest user, Sim can simulate a $(\textsc{Garble}, sid, ssid)$ message itself. The garbled circuit F and the decoding information d are calculated in the same way in both worlds, using $\mathrm{Gb}(1^\lambda, C)$. The only difference is the encoded key K. In the ideal world, K is an encoding of a random value k, which is chosen for the honest server S by Sim. In the real world, K is an encoding of the PRF-key k of that server. However, in both cases, k is a uniformly random value in $\{0,1\}^m$ and in both experiments, k is encoded via Enc. Therefore, the two experiments are distributed identically.

Responses of the Random Oracles:

- $H_1(\cdot)$ queries:

  In the real world, a random oracle chooses a uniformly random output for every fresh query and stores this random value as "hash" of the input. On further queries, that stored value is returned. The simulator answers the calls to H_1 exactly, as a real random oracle would do, with uniformly random values $h \in \{0,1\}^n$.

- $H_2(\cdot,\cdot)$ queries:

  In the following, we will only argue why the simulated H_2 is indistinguishable from the original H_2 in the real execution. As we've seen at the beginning of Section 3.5, the random oracle H_2 must also be compared to the user's output. We defer this discussion to the next paragraph. We distinguish the following cases:

Case 1: There is no record $\langle H_1, p, h\rangle$ found: The random oracle is programmed with a uniformly random value. In this case, Sim behaves like the real random oracle.

Case 2: Records $\langle H_1, p, h\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exist, such that $\mathsf{De}(d, \mathsf{Ev}(F, X \parallel K)) = y$: In that case, the value y was calculated with the garbled circuit of an honest server, with overwhelming probability. That means the simulator can query $\mathcal{F}_{\mathrm{OPRF}}$ for the correct output value by choosing an unused subsession id $ssid'$ and calling $(\textsc{Eval}, sid, ssid', p)$ and $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$. If the ideal functionality does not answer, Sim aborts. Remember that $\mathcal{F}_{\mathrm{OPRF}}$ does only ignore $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ messages in one of the following three cases:

  a) There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$.

  b) $i = \mathsf{S}$ but $\mathsf{tx}(\mathsf{S}) = 0$.

  c) S is honest but $i \neq \mathsf{S}$.

  We prove in Lemma 1 that the condition in Item 2 happens at most with negligible probability. Further, the first and the third abort condition Item 1 and Item 3 can not occur in this case, as Sim itself sends the message $(\textsc{Eval}, sid, ssid', \mathsf{S}, p)$ to $\mathcal{F}_{\mathrm{OPRF}}$ just before sending $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$.

  $H_2(p, y)$ is then programmed to the output ρ of $\mathcal{F}_{\mathrm{OPRF}}$. This is, by the definition of $\mathcal{F}_{\mathrm{OPRF}}$, a uniformly random value.

Case 3: There is a record $\langle H_1, p, h\rangle$ but no record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exists, such that $\mathsf{De}(d, \mathsf{Ev}(F, X \parallel K)) = y$: In that case, Sim checks the keys of all corrupted parties $k_{\hat{\mathsf{S}}}$. Note that Sim knows those keys as we assume static corruption only and that the adversary learns all the randomness of a corrupted party. If there is such a corrupted server with key k_S such that $C_{k_S}(h) = y$, the simulator can use its ability to offline evaluate PRFs from corrupted parties. Thus, Sim will program $H_2(p, y)$ to the output of the offline evaluation. This will, again, be a uniformly random value $\rho \in \{0,1\}^l$. If no such key exists $H_2(p, y)$ is set to a uniformly random value, as from a real random oracle.

Honest User Output $(\textsc{EvalOut}, sid, ssid, \rho)$ from U as response to an $(\textsc{Eval}, sid, ssid, \mathsf{S}, pw)$ message:

In the real world, ρ is calculated as $\rho = H_2(p, \mathsf{De}(d, \mathsf{Ev}(F, X \parallel K)))$, where (F, K, d) was generated by the server and X are the labels received via OT for $x = H_1(p)$. In the ideal world, ρ is chosen uniformly at random by $\mathcal{F}_{\mathrm{OPRF}}$ if a fresh $(\textsc{Eval}, sid, ssid, \mathsf{S}, p)$ message was sent. Remember that $\mathcal{F}_{\mathrm{OPRF}}$ keeps an internal table $T_{sid}(i, \cdot)$ for possible server IDs i. If an honest user with input p interacts with S, the functionality $\mathcal{F}_{\mathrm{OPRF}}$ will send $\rho = T_{sid}(\mathsf{S}, p)$ as output for the honest user. The simulator must produce the same output ρ for $H_2(p, y)$ if $y = C_k(H_1(p))$ holds for S's key k. We therefore have to compare the output of H_2 with the outputs of $\mathcal{F}_{\mathrm{OPRF}}$. We distinguish the following cases in simulation of H_2:

Case 1: There is no record $\langle H_1, p, h\rangle$ found: Sim only needs to program the random oracle, if p and y do occur in a protocol execution. More precisely, if $y = C_k(H_1(p))$ holds for some server's key k. That is because in this case $\mathcal{F}_{\mathrm{OPRF}}$ can eventually output a value ρ as the output of an honest user with input p interacting with a server with key k. In other words, if there is a server with key k such that k "maps" $H_1(p)$ to y, then there can be a protocol execution that leads to a query $H_2(p, y)$ where Sim must program H_2. We will call a query (p, y) *relevant* if there is a server with key k, such that $y = C_k(H_1(p))$. In the following, we bound the probability for the event that (p, y) becomes relevant, when $H_1(p)$ is not determined yet.

Let $t \in \mathbb{N}$ be the number of servers in the protocol execution. Let $k_1, \dots, k_t$ be the keys used by the servers and let $n \in \Omega(\lambda)$ be the output length of C. We assumed in the beginning that $C_{k_i}(\cdot)$ is a permutation for every $i \in \{1, \dots, t\}$. Thus, if we choose some uniformly random input $x \in \{0,1\}^n$, we get that $C_{k_i}(x) \in \{0,1\}^n$ is uniformly random. If $H_1(p)$ is not queried yet, we have for every $i \in \{1, \dots, t\}$ and every $y \in \{0,1\}^n$:

$$\Pr[C_{k_i}(H_1(p)) = y] \le \frac{1}{2^n},$$

where the probability is taken over the random output of H_1. This follows from the fact that $C_{k_i}(\cdot)$ is a permutation.

We have for every tuple $(p, y) \in \{0,1\}^* \times \{0,1\}^n$ where $H_1(p)$ was not queried yet:

$$\begin{aligned}\Pr[(p, y) \text{ becomes relevant}] &= \Pr\left[\bigvee_{i=1}^{t} (C_{k_i}(H_1(p)) = y)\right] \\ &\le \sum_{i=1}^{t} \Pr[C_{k_i}(H_1(p)) = y] \\ &= t \Pr[C_{k_1}(H_1(p)) = y] \\ &\le \frac{t}{2^n},\end{aligned}$$

where the probability is taken over the randomness of $H_1(p)$. As t is polynomial in λ and we assume $n \in \Omega(\lambda)$, a tuple (p, y) becomes relevant at most with negligible probability if $H_1(p)$ was not queried yet. Thus, Sim can assign a uniformly random value to $H_2(p, y)$.

Case 2: Records $\langle H_1, p, h\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exist, such that $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \parallel K)) = y$:

In this case, the value h is the output of the random oracle H_1 on input p. The tuple (p, y) is relevant, because the key of an honest server produces the output y, when the input h is provided to the circuit. Sim knows to which server the key belongs, as the record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ explicitly contains the server id S. The simulator Sim sends $(\textsc{Eval}, sid, ssid', \mathsf{S}, p)$ to $\mathcal{F}_{\mathrm{OPRF}}$ for a new subsession id $ssid'$. That means, Sim initiates a new protocol execution and requests itself the output value $\rho = T_{sid}(\mathsf{S}, p)$ from $\mathcal{F}_{\mathrm{OPRF}}$. Next, Sim can safely send the $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ message, without decreasing the ticket counter of S below 0. Intuitively, this is because the key of an honest server and the input labels of an honest user are hidden from $\mathcal{E}$. We prove that in Lemma 1. The random oracle $H_2(p, y)$ is programmed to the answer ρ of $\mathcal{F}_{\mathrm{OPRF}}$. The programming ensures that $\mathcal{E}$ will get the same output $\rho = H_2(p, y)$ when invoking an execution of the protocol between a honest user with input p and the honest server that generated (F, K, d).

Case 3: There is a record $\langle H_1, p, h\rangle$ but no record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exists, such that $\mathsf{De}(d, \mathsf{Ev}(F, X \parallel K)) = y$:

In that case, the value h is the output of the random oracle H_1 on input p, but no honest server key maps h to $y = C_k(h)$. Thus, Sim checks the keys of all corrupted server $k_{\hat{\mathsf{S}}}$. If one of the keys $k_{\hat{\mathsf{S}}}$ is such that $C_{k_{\hat{\mathsf{S}}}}(h) = y$ holds, Sim will use its ability to offline evaluate corrupted server's tables $T_{sid}(\hat{\mathsf{S}}, \cdot)$. The simulator Sim sends $(\textsc{OfflineEval}, sid, \hat{\mathsf{S}}, p)$ to $\mathcal{F}_{\mathrm{OPRF}}$ and receives the answer $(\textsc{OfflineEval}, sid, \rho)$ from $\mathcal{F}_{\mathrm{OPRF}}$. Note, that Sim will always receive an answer in this case, as $\hat{\mathsf{S}}$ is the identity of a corrupted server.

Sim programs $H_2(p, y)$ to the output ρ of the offline evaluation. $\mathcal{E}$ will get the same ρ as output from an execution of the protocol between a user with input p and the corrupted server with key $k_{\hat{\mathsf{S}}}$.

If there are multiple such keys, i.e., the condition in line 101 of Figure 3.8 is true, Sim aborts. This happens at most with negligible probability, as we prove in Lemma 2.

If no such key exists $H_2(p, y)$ is set to a uniformly random value, as in this case y does not correspond to some protocol execution, i.e., (p, y) is not relevant. □

Lemma 1. *Let the garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ have* privacy*, as defined in Definition 11. When interacting with the simulator in Figures 3.5 to 3.8, for each server* S *the probability that a $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ message for $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$ is sent when the ideal functionality's ticket counter* $\mathsf{tx}(\mathsf{S})$ *is 0, is negligible. That means, only with negligible probability $\mathcal{F}_{\mathrm{OPRF}}$ ignores a* RcvCmplt *message because the ticket counter is 0.*

Proof. The ticket counter $\mathrm{tx}(\mathrm{S})$ is only increased by $(\textsc{SndrComplete}, sid, ssid)$ messages from S to $\mathcal{F}_{\mathrm{OPRF}}$, i.e., by invocations of the server by $\mathcal{E}$. The counter is decreased by $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ messages from Sim to $\mathcal{F}_{\mathrm{OPRF}}$. The simulator from Figures 3.5 to 3.8 sends $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ messages in two cases. We will regard them separately:

Case 1: Sim received a query $H_2(p, y)$ and has records $\langle H_1, p, h\rangle$ and $\langle \mathrm{S}, sid, ssid, (F, K, d), X[0], X[1]\rangle$ such that $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \parallel K)) = y$, i.e., the condition in line 85 in Figure 3.8 is true:

As Sim found the record $\langle \mathrm{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$, we can be sure that a $(\textsc{SndrComplete}, sid, ssid)$ messages was sent by $\mathcal{E}$ to Sim. This means the counter $\mathrm{tx}(\mathrm{S})$ was increased at least once before the circuit was garbled. This holds, because Sim does only store the record $\langle \mathrm{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ when it received a $(\textsc{SndrComplete}, sid, ssid)$ message from $\mathcal{F}_{\mathrm{OPRF}}$.

Next, we know that $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \parallel K)) = y$ holds. If that holds, Sim can safely assume that the server S that created (F, K, d) is the server for which Sim must query an OPRF output $\rho = T_{sid}(\mathrm{S}, h)$ value from $\mathcal{F}_{\mathrm{OPRF}}$. We argue in Lemma 2 that another key $k' \neq k$ could lead to the same result y with at most negligible probability.

The $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ messages in line 74 of Figure 3.7 are only sent to produce an output of honest users. If the user is corrupted, that implies that there cannot be a message $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ produced by Sim in response to an $(\text{OT-}\textsc{Received}, (ssid, i))$ message, in line 74 of Figure 3.7.

If the user is honest, we show in Lemma 3 that the situation we currently argue about, i.e., Sim received a query $H_2(p, y)$ and has records $\langle H_1, p, h\rangle$ and $\langle \mathrm{S}, sid, ssid, (F, K, d), X[0], X[1]\rangle$ such that $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \parallel K)) = y$, happens at most with negligible probability.

In conclusion, sending $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathrm{S})$ in line 89 of Figure 3.8 as a consequence of a $H_2(p, y)$ query will decrease the ideal functionality's counter $\mathrm{tx}(\mathrm{S})$ by one. Another $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ is sent in line line 74 of Figure 3.7 at most with negligible probability. Querying the same tuple $H_2(p, y)$ again won't result in a second $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathrm{S})$ message in line 89 of Figure 3.8, as the output of $H_2(p, y)$ is already defined. Thus, the counter is only decreased by one if it was increased at least by one before with a $(\textsc{SndrComplete}, sid, ssid)$ message to $\mathcal{F}_{\mathrm{OPRF}}$.

Case 2: Sim received all n messages $(\text{OT-}\textsc{Received}, (ssid, i))$ a garbling (F, K, d) for a subsession $ssid$, where all the recorded OT-requests x_i are $\neq \perp$, i.e., the condition in line 71–73 of Figure 3.7 is true:

We know that the user already received a garbling (F, K, d), as either the clause in line 72 or the clause in line 73 of Figure 3.7 is true. We assume passive adversaries, which implies that a $(\textsc{SndrComplete}, sid, ssid)$ message was already sent to $\mathcal{F}_{\mathrm{OPRF}}$. Else, the server would not have created the garbling (F, K, d). This means, the counter $\mathrm{tx}(\mathrm{S})$ is only decreased by one with a $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathrm{S})$ message in line 74 of Figure 3.7 if it is increased at least once before by a $(\textsc{SndrComplete}, sid, ssid)$ message to $\mathcal{F}_{\mathrm{OPRF}}$.

We argue why there cannot be another (RcvCmplt, *sid*, *ssid*, P, S) message with $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$ for the same *sid*, *ssid* and label S. There cannot be another (RcvCmplt, *sid*, *ssid*, U, S) message sent in line line 74 of Figure 3.7 for the same subsession *ssid*. This holds, because we argue about the case where Sim simulates the behavior of an honest user. Sim only sends (RcvCmplt, *sid*, *ssid*, U, S) once, at the moment when all n input labels are received by the user. If Sim receives further labels for the same *ssid* that will not trigger a second (RcvCmplt, *sid*, *ssid*, U, S) message for this *ssid*. Remember that there are only two situations in which Sim sends (RcvCmplt, *sid*, *ssid*, P, S) with $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$. The first one is the situation where all n messages (OT-Received, $(ssid, i)$) were received. This is the situation we currently reason about. The second one is when an $H_2(p, y)$ is received and it turns out that the corresponding key k^* belongs to an honest server. We argue why the second situation can happen at most with negligible probability. All recorded OT-requests x_i are $\neq \perp$. Thus, the corresponding (OT-Receive, $(ssid, i)$) messages were simulated by Sim for an honest user. But the (RcvCmplt, *sid*, *ssid'*, $\mathcal{A}$, S) messages in line 89 of Figure 3.8 are only sent if the subsession is executed with an honest server. Again, it follows from Lemma 3 that the (RcvCmplt, *sid*, *ssid'*, $\mathcal{A}$, S) messages in line 89 of Figure 3.8 is sent at most with negligible probability. □

Lemma 2. *For $m, n, l \in \Omega(\lambda)$ let the function $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^l$ be PRF. Let $t \in \mathbb{N}$ be polynomial in λ. For every $x \in \{0,1\}^n$ and uniformly random and independently drawn keys $k_1, \dots, k_t \in \{0,1\}^m$, there are at most with negligible probability in λ indices $i, j \in \{1, \dots, t\}$ with $i \neq j$ such that $\mathsf{F}_{k_i}(x) = \mathsf{F}_{k_j}(x)$.*

Proof. We start with the simpler case that the first index is $i = 1$. In other words, we bound the probability that there is a key in $k_2, \dots, k_t$, such that $\mathsf{F}_{k_1}(x) = \mathsf{F}_{k_j}(x)$. For $x \in \{0,1\}^n$, we consider the following sequence of hybrid experiments: In the first experiment E_1, the experiment chooses uniformly random keys $k_2, \dots, k_t \in \{0,1\}^m$ and outputs 1 iff there is one $j \in \{2, \dots, t\}$ such that $\mathsf{F}_{k_1}(x) = \mathsf{F}_{k_j}(x)$. E_2 is defined as above, except that the second value $\mathsf{F}_{k_2}(x)$ is replaced by a uniformly random value $y_2 \in \{0,1\}^l$. Now, for every $r \in \{3, \dots t\}$, we define the experiments E_r as follows: The experiment chooses uniformly random values $y_2, \dots, y_r \in \{0,1\}^l$ and uniformly random keys $k_{r+1}, \dots, k_t \in \{0,1\}^m$. The experiment outputs 1 iff $\mathsf{F}_{k_1}(x) = y_j$ for $j \in \{2, \dots, r\}$ or $\mathsf{F}_{k_1}(x) = \mathsf{F}_{k_j}(x)$ for $j \in \{r+1, \dots, t\}$. Finally, E_t is the experiment, where all values $y_2, \dots, y_t$ are uniformly random. We get by a union-bound that E_t outputs 1 with probability

$$\Pr[E_t = 1] = \Pr\left[\bigvee_{j=2}^{t} (y_j = \mathsf{F}_{k_1}(x))\right] \leq \frac{(t-1)}{2^l},$$

where the probability is taken over the random choices of $y_2, \dots, y_t$. Note, that $\mathsf{F}_{k_1}(x)$ is constant here. Assume, by way of contradiction, that the probability that experiment E_1 outputs 1 with a noticeable probability. Then there is an index $N \in \{1, \dots, t-1\}$ such that the difference $\Delta := |\Pr[E_N = 1] - \Pr[E_{N+1} = 1]|$ is noticeable. We construct a distinguisher $\mathcal{D}$ for the PRF security experiement, see Definition 1. $\mathcal{D}$ proceeds as the experiment E_N but instead of using k_N to calculate $\mathsf{F}_{k_N}(x)$, the distinguisher $\mathcal{D}$ queries x from its PRF oracle and receives an output y^*. If the oracle answers with a PRF output y^*, the output of $\mathcal{D}$ is exactly distributed as in E_N. If the oracle answers with a truly random output, the output of $\mathcal{D}$ is distributed as in E_{N+1}. Thus, by our assumption, $\mathcal{D}$ has a noticble advantage Δ in the PRF experiment, which is a contradiction to F being a PRF. This concludes the hybrid argument.

Above, we bound the probability that there is another key whose output collides with k_1. With a completely analogous reduction, we get a similar inequality for every $k_1, \dots, k_t$. Hence, we have for all $x \in \{0,1\}^n$ that the probability that there are $i, j \in \{1, \dots, t\}$ with $i \neq j$ such that $\mathsf{F}_{k_i}(x) = \mathsf{F}_{k_j}(x)$ is

$$\Pr\left[\bigvee_{i=1}^{t}\left(\bigvee_{j=1; j\neq i}^{t} \mathsf{F}_{k_i}(x) = \mathsf{F}_{k_i}(x)\right)\right] \leq \frac{t(t-1)}{2^l} + \mathrm{negl}(\lambda).$$

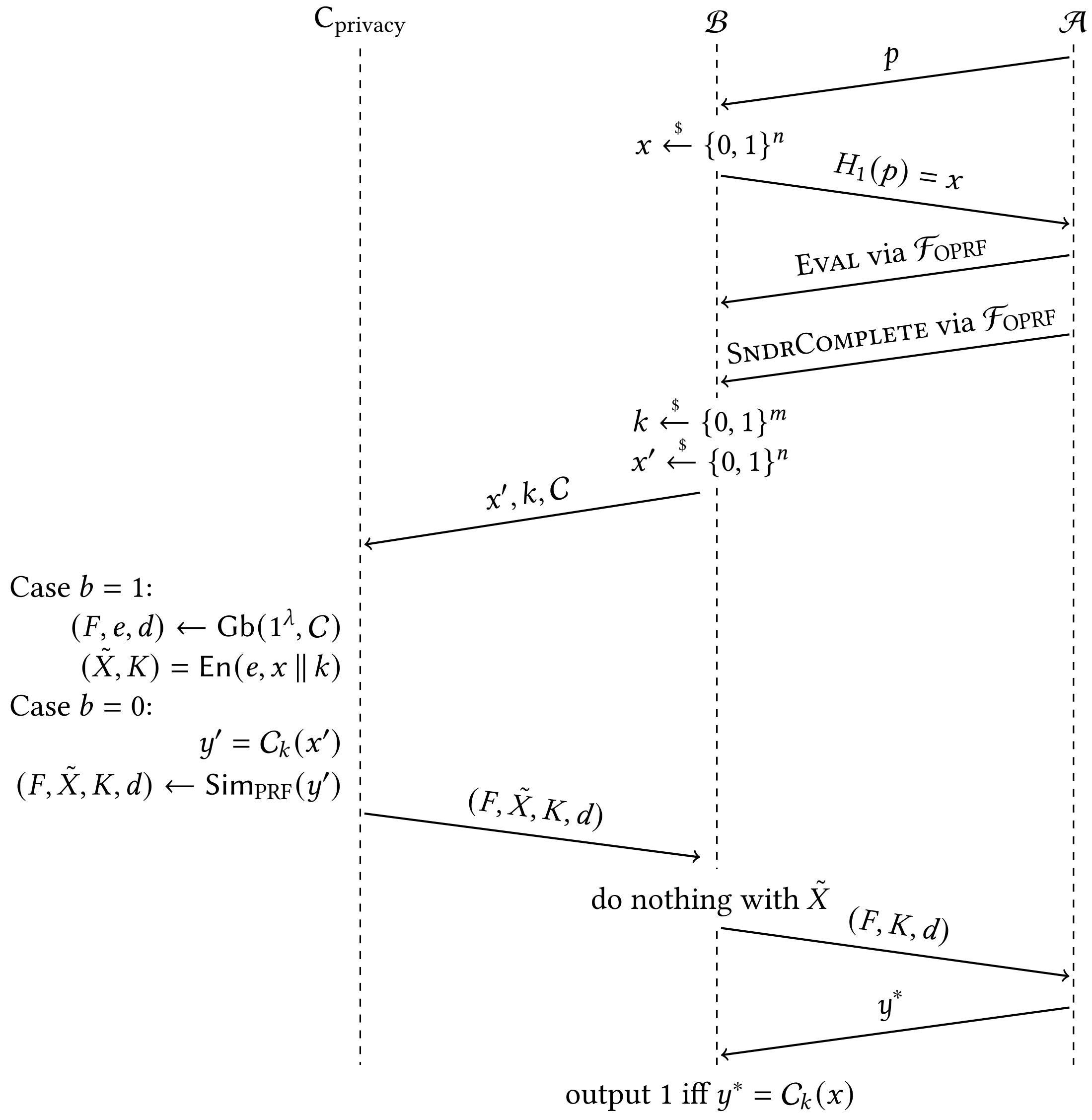
Lemma 3. *Let the garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ have privacy, as defined in Definition 11. Let C be the boolean circuit of a PRF, as defined in Definition 1. Suppose the adversary $\mathcal{A}$ initiates an OPRF execution between an honest server and an honest user with input p. The adversary $\mathcal{A}$ can at most with negligible probability send a request $H_2(p, y) \in \{0,1\}^n$ such that $C_k(H_1(p)) = y$, where $k \in \{0,1\}^m$ is the key of the honest server.*

Proof. Without loss of generality, we can assume that $\mathcal{A}$ requested $h = H_1(p)$ for the user input $p \in \{0,1\}^*$. Further, we assume that $\mathcal{A}$ received a garbled circuit, an encoded key , and decoding information (F, K, d), that were created by Sim in simulating an honest server. We know, $\mathcal{A}$ received no labels for the user input h, as Sim simulated the OT for an honest user.

Assume, by way of contradiction, that $\mathcal{A}$ calculates output $y \in \{0,1\}^n$ such that $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \,\|\, K)) = y$ with noticeable probability P. First, we construct an adversary $\mathcal{B}$ that plays the privacy experiment as in Figure 2.9 and communicates with $\mathcal{A}$ as if $\mathcal{B}$ was the simulator. As we assume that the garbling scheme has privacy, we will get the existence of a simulator $\mathsf{Sim}_{\mathrm{PRF}}$ for the privacy experiment. We will use this simulator $\mathsf{Sim}_{\mathrm{PRF}}$ and the adversary $\mathcal{B}$ to construct a second adversary $\mathcal{B}_{\mathrm{PRF}}$ that will have noticeable success probability in distinguishing the PRF C from a truly random function, which is a contradiction to our assumption that C satisfies Definition 1 of a PRF.

$\mathcal{B}$ plays the UC-security experiment with $\mathcal{A}$. Let $t \in \mathbb{N}$ be the number of subsessions that $\mathcal{A}$ invokes between an honest server and an honest user. The adversary $\mathcal{B}$ initially chooses an index $i^* \in \{1, \dots, t\}$ uniformly at random. $\mathcal{B}$ behaves like our normal simulator from Figures 3.5 to 3.8, except when $\mathcal{A}$ initiates a subsession between an honest server and an honest user. If that session is the i^*th of those sessions, $\mathcal{B}$ behaves as follows: When $\mathcal{B}$ receives an $(\textsc{Eval}, sid, ssid, \mathsf{U}, \mathsf{S})$ message and a $(\textsc{SndrComplete}, sid, ssid, \mathsf{S})$ message from $\mathcal{F}_{\mathrm{OPRF}}$, $\mathcal{B}$ must simulate the honest server. It chooses a uniformly random key $k \in \{0,1\}^m$ and a uniformly random value $x' \in \{0,1\}^n$. The second value x' can be seen as a "mock" input to the privacy challenger $\mathsf{C}_{\mathrm{privacy}}$. Note that x' and the actual hash value $x = H_1(p)$ are chosen independently. $\mathcal{B}$ answers queries to $H_1(p)$ as usual by choosing $x \in \{0,1\}^n$ uniformly at random and storing $\langle H_1, p, x\rangle$. The adversary $\mathcal{B}$ sends

(x', k, C) to $\mathsf{C}_{\text{privacy}}$. The privacy challenger chooses $b \in \{0,1\}$ uniformly at random. If $b = 1$, it calculates $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$ and $(\tilde{X}, K) = \mathsf{En}(e, x' \,\|\, k)$. If $b = 0$, it calculates $y' = \mathsf{ev}(C, x' \,\|\, k) = C_k(x')$. Next, $\mathsf{C}_{\text{privacy}}$ runs the simulator $\mathsf{Sim}_{\mathsf{PRF}}$ on input y'. The simulator $\mathsf{Sim}_{\mathsf{PRF}}$ outputs $(F, \tilde{X}, K, d)$. In both cases $b = 1$ and $b = 0$, the challenger $\mathsf{C}_{\text{privacy}}$ sends $(F, \tilde{X}, K, d)$ to $\mathcal{B}$. Now, $\mathcal{B}$ uses this garbled circuit to simulate the honest server. That means, $\mathcal{B}$ sends (F, K, d) to $\mathcal{A}$, formatted as if U sent it to S via $\mathcal{F}_{\mathrm{AUTH}}$. Note that $\tilde{X}$ is *not* sent to $\mathcal{A}$ as our actual OPRF simulator from Figures 3.5 to 3.8 would also not do that. Finally, $\mathcal{B}$ checks for ever H_2 query (p, y^*) from $\mathcal{A}$, if $y^* = C_k(H_1(p))$ holds. Only if that is the case, $\mathcal{B}$ outputs 1, else it outputs 0. We depicted the reduction in Figure 3.3.



$\mathsf{C}_{\text{privacy}}$
$\mathcal{B}$
$\mathcal{A}$
p
$x \xleftarrow{\$} \{0,1\}^n$
$H_1(p) = x$
Eval via $\mathcal{F}_{\mathrm{OPRF}}$
SndrComplete via $\mathcal{F}_{\mathrm{OPRF}}$
$k \xleftarrow{\$} \{0,1\}^m$
$x' \xleftarrow{\$} \{0,1\}^n$
x', k, C
Case $b = 1$:
$(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$
$(\tilde{X}, K) = \mathsf{En}(e, x \,\|\, k)$
Case $b = 0$:
$y' = C_k(x')$
$(F, \tilde{X}, K, d) \leftarrow \mathsf{Sim}_{\mathsf{PRF}}(y')$
$(F, \tilde{X}, K, d)$
do nothing with $\tilde{X}$
(F, K, d)
y^*
output 1 iff $y^* = C_k(x)$

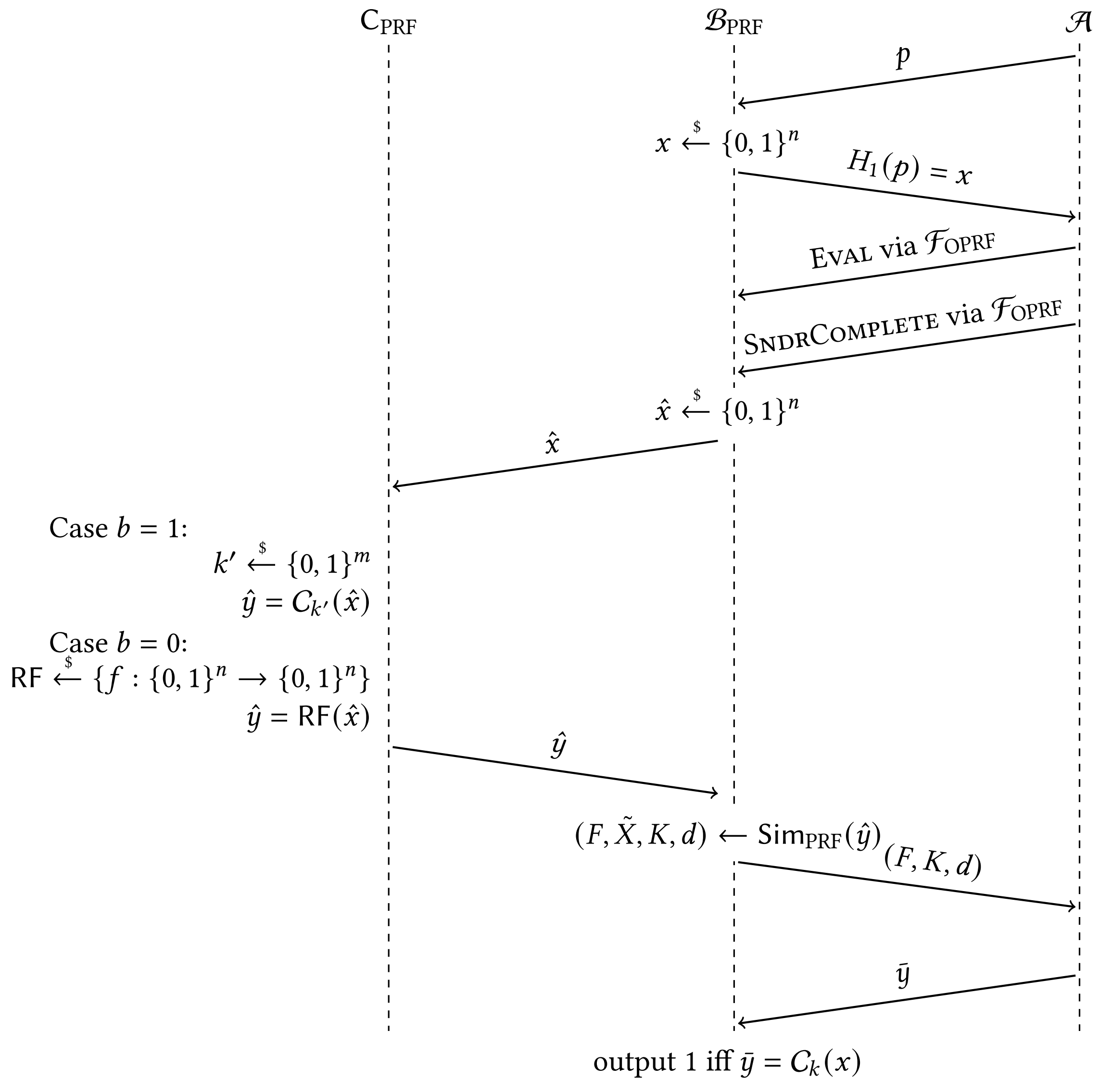
Figure 3.3.: Reduction on the Privacy Property of the Garbling Scheme.

In the case where the challenger $\mathsf{C}_{\text{privacy}}$ chose $b = 1$, the view of $\mathcal{A}$ is identically distributed as in a normal OPRF execution with our simulator Sim. That holds, because $k \in \{0,1\}^m$ is also chosen uniformly at random and F and d are also calculated as $(F, e, d) \leftarrow$

$\mathsf{Gb}(1^\lambda, C)$. The calculation of those values is completely independent of the value x'. The encoded key is calculated as $(\tilde{X}, K) = \mathsf{En}(e, x' \parallel k)$, but the value of K does only depend on e and not on x'. With probability $1/t$, the adversary $\mathcal{B}$ chooses the right index i^* of the execution, where $\mathcal{A}$ succeeds in calculating y^* such that $y^* = C_k(H_1(p))$ holds. By our assumption, this means that $\mathcal{B}$ outputs 1 with probability P/t, which is noticeable. Now, the privacy of the garbling scheme guarantees us that a simulator $\mathsf{Sim}_{\mathrm{PRF}}$ exists that makes $\mathcal{B}$ output 1 with noticeable probability P' in the case $b = 0$. We now show in a second reduction that we can build an adversary $\mathcal{B}_{\mathrm{PRF}}$ that uses $\mathsf{Sim}_{\mathrm{PRF}}$ and $\mathcal{A}$ as subroutines and that distinguishes between a PRF and a truly random function with noticeable probability.

Like $\mathcal{B}$ above, the adversary $\mathcal{B}_{\mathrm{PRF}}$ plays the UC-security experiment with $\mathcal{A}$. The adversary $\mathcal{B}_{\mathrm{PRF}}$ chooses an index $i^* \in \{1, \dots, t\}$ uniformly at random, where $t \in \mathbb{N}$ is the number of subsession of honest users with honest servers. For the i^*th subsession, when $\mathcal{B}_{\mathrm{PRF}}$ receives an $(\textsc{Eval}, sid, ssid, \mathrm{U}, \mathrm{S})$ message and a $(\textsc{SndrComplete}, sid, ssid, \mathrm{S})$ message from $\mathcal{F}_{\mathrm{OPRF}}$, the adversary $\mathcal{B}_{\mathrm{PRF}}$ must simulate the honest server. $\mathcal{B}_{\mathrm{PRF}}$ chooses a uniformly random value $\hat{x} \in \{0,1\}^n$ and sends $\hat{x}$ to to PRF challenger $\mathrm{C}_{\mathrm{PRF}}$. The challenger $\mathrm{C}_{\mathrm{PRF}}$ chooses a bit $b' \in \{0,1\}$ uniformly at random. If $b' = 1$, the challenger calculates $\hat{y} = C_{k'}(\hat{x})$, for some uniformly random $k' \in \{0,1\}^m$. If $b' = 0$, the challenger $\mathrm{C}_{\mathrm{PRF}}$ sets $\hat{y} = \mathsf{RF}(\hat{x})$ where $\mathsf{RF} \in \{f : \{0,1\}^n \to \{0,1\}^n\}$ is chosen uniformly at random. $\mathrm{C}_{\mathrm{PRF}}$ sends $\hat{y}$ to $\mathcal{B}_{\mathrm{PRF}}$. The adversary $\mathcal{B}_{\mathrm{PRF}}$ calls $\mathsf{Sim}_{\mathrm{PRF}}$ on input $\hat{y}$ and receives $(F, \tilde{X}, K, d)$ as output. $\mathcal{B}_{\mathrm{PRF}}$ simulates a message to $\mathcal{A}$ as if the honest user sent (F, K, d) to the honest server via $\mathcal{F}_{\mathrm{AUTH}}$. The adversary $\mathcal{A}$ answers with a value $\bar{y}$. Now, $\mathcal{B}_{\mathrm{PRF}}$ checks for every H_2 query $(p, \bar{y})$ if $\bar{y} = C_k(H_1(p))$ holds. Only if that is true, $\mathcal{B}_{\mathrm{PRF}}$ outputs 1, else it outputs 0. We depicted the reduction in Figure 3.4.

Suppose that $\mathcal{B}_{\mathrm{PRF}}$ chose the correct index i^*, i.e., the subsession in which $\mathcal{A}$ is successful in sending the query $(p, \bar{y})$. That happens with probability $1/t$. In case $b' = 1$, the view of $\mathsf{Sim}_{\mathrm{PRF}}$ is exactly distributed as in the privacy experiment with $\mathcal{B}$ above. By our assumption on $\mathsf{Sim}_{\mathrm{PRF}}$, the environment $\mathcal{A}$ has noticeable probability P' to send a query $(p, \bar{y})$ such that $\bar{y} = C_k(H_1(p))$. That means, the overall success probability of $\mathcal{B}_{\mathrm{PRF}}$ in this case is P'/t, which is noticeable. In case $b' = 0$, the value $\hat{y} \in \{0,1\}^n$ is uniformly random. That means in particular that $\mathsf{Sim}_{\mathrm{PRF}}$'s output $(F, \tilde{X}, K, d)$ is stochastically independent of $C_k(H_1(p))$. In that case, the input (F, K, d) gives $\mathcal{A}$ information-theoretically no advantage in guessing $C_k(H_1(p))$. Consequently, $\mathcal{A}$ outputs $(p, \bar{y})$ such that $\bar{y} = C_k(H_1(p))$ at most with probability 2^{-n}. This is a contradiction to the PRF probability, as $\mathcal{B}_{\mathrm{PRF}}$ outputs 1 with noticeable probability in the case $b' = 1$. In conclusion, no such simulator $\mathsf{Sim}_{\mathrm{PRF}}$ can exists, which is a contradiction to the assumed privacy of the garbling scheme. Thus, the assumed adversary $\mathcal{A}$ cannot exist. $\square$



C_{PRF}
$\mathcal{B}_{\text{PRF}}$
$\mathcal{A}$
p
$x \xleftarrow{\$} \{0,1\}^n$
$H_1(p) = x$
Eval via $\mathcal{F}_{\text{OPRF}}$
SndrComplete via $\mathcal{F}_{\text{OPRF}}$
$\hat{x} \xleftarrow{\$} \{0,1\}^n$
$\hat{x}$
Case $b = 1$:
$k' \xleftarrow{\$} \{0,1\}^m$
$\hat{y} = C_{k'}(\hat{x})$
Case $b = 0$:
$\mathsf{RF} \xleftarrow{\$} \{f : \{0,1\}^n \to \{0,1\}^n\}$
$\hat{y} = \mathsf{RF}(\hat{x})$
$\hat{y}$
$(F, \tilde{X}, K, d) \leftarrow \mathsf{Sim}_{\text{PRF}}(\hat{y})$
(F, K, d)
$\bar{y}$
output 1 iff $\bar{y} = C_k(x)$

Figure 3.4.: Reduction on the PRF Property.

```
Initialization
for all corrupted servers $\hat{S}$ with key $k_{\hat{S}}$ :
    record $\langle k_{\hat{S}}, \hat{S} \rangle$

On $(\textsc{Init}, sid, \mathrm{S})$ from $\mathcal{F}_{\mathrm{OPRF}}$
If this is the first $(\textsc{Init}, \mathrm{S}, sid)$ message from $\mathcal{F}_{\mathrm{OPRF}}$
$k \xleftarrow{\$} \{0,1\}^m$; record $\langle \mathrm{S}, sid, k \rangle$

On $(\textsc{Eval}, sid, ssid, \mathrm{U}, \mathrm{S})$ from $\mathcal{F}_{\mathrm{OPRF}}$
// simulate sending $(\textsc{Garble}, sid, ssid)$ on behalf of U to S via $\mathcal{F}_{\mathrm{AUTH}}$
send $(\textsc{Sent}, mid, \mathrm{U}, \mathrm{S}, (\textsc{Garble}, sid, ssid))$ to $\mathcal{A}$.
record $\langle \textsc{Garble}, sid, ssid \rangle$
on $(\textsc{ok}, mid)$ from $\mathcal{A}$ if S is corrupted :
    send $(\textsc{Sent}, mid, \mathrm{U}, \mathrm{S}, (\textsc{Garble}, sid, ssid))$ to S
if U is honest and $\exists \langle \textsc{SndrComplete}, sid, ssid \rangle$ :
    goto label SimulateGarbling

On $(\textsc{SndrComplete}, sid, ssid, \mathrm{S})$ from $\mathcal{F}_{\mathrm{OPRF}}$
if U is corrupted and $\nexists \langle \text{received}\textsc{Garble}, sid, ssid \rangle$ :
    record $\langle \textsc{SndrComplete}, sid, ssid \rangle$
elseif U is honest and $\nexists \langle \textsc{Garble}, sid, ssid \rangle$ :
    record $\langle \textsc{SndrComplete}, sid, ssid \rangle$
else
    SimulateGarbling :
    // simulate receiving $(\textsc{Garble}, sid, ssid)$ from U via $\mathcal{F}_{\mathrm{AUTH}}$
    send $(\textsc{Sent}, mid, \hat{\mathrm{U}}, \mathrm{S}, (\textsc{Garble}, sid, ssid))$ to $\mathcal{A}$.
    on $(\textsc{ok}, mid)$ from $\mathcal{A}$ :
        search for recorded tuple $\langle \mathrm{S}, sid, k \rangle$
        if $\nexists \langle \mathrm{S}, sid, k \rangle$ :
            $k \xleftarrow{\$} \{0,1\}^m$; record $\langle \mathrm{S}, sid, k \rangle$
        $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$
        $(X[0^n] \,\|\, K) := \mathsf{En}(e, 0^n \,\|\, K)$; $(X[1^n] \,\|\, K) := \mathsf{En}(e, 1^n \,\|\, K)$
        // simulate sending (F,K,d) from S to $\hat{\mathrm{U}}$ via $\mathcal{F}_{\mathrm{AUTH}}$
        send $(\textsc{Sent}, mid, \mathrm{S}, \hat{\mathrm{U}}, (sid, ssid, (F, K, d)))$ to $\mathcal{A}$
        on $(\textsc{ok}, sid, ssid)$ from $\mathcal{A}$ :
            send $(\textsc{Sent}, mid, \mathrm{S}, \hat{\mathrm{U}}, (sid, ssid, (F, K, d)))$ to $\hat{\mathrm{U}}$
        record $\langle \mathrm{S}, sid, ssid, (F, K, d), X[0^n], X[1^n] \rangle$
        // simulate sending labels $X[0], X[1]$ via $\mathcal{F}_{\mathrm{OT}}$.
        for $i = 1, \ldots, n$ :
            record $\langle (ssid, i), (X_i[0], X_i[1]) \rangle$
            send $(\text{OT-}\textsc{Send}, (ssid, i))$ to $\mathcal{A}$
```

Figure 3.5.: The Simulator Sim Part I. Simulation of Messages From $\mathcal{F}_{\mathrm{OPRF}}$.

```
On (SEND, $mid$, S, (GARBLE, $sid$, $ssid$)) from $\mathcal{A}$ on behalf of $\hat{\mathrm{U}}$

if $\nexists\langle$SNDRCOMPLETE, $sid$, $ssid\rangle$ :
    record $\langle$receivedGARBLE, $sid$, $ssid\rangle$
else
    goto label SimulateGarbling

On (SEND, $mid$, U, ($sid$, $ssid$, $(F, K, d)$)) from $\mathcal{A}$ on behalf of $\hat{\mathrm{S}}$ to U

// Simulator gets message $(F, K, d)$ from $\hat{\mathrm{S}}$ to U via $\mathcal{F}_{\mathrm{AUTH}}$
send (SENT, $mid$, $\hat{\mathrm{S}}$, U, ($sid$, $ssid$, $(F, K, d)$)) to $\mathcal{A}$
on (OK, $mid$) from $\mathcal{A}$ :
    if $\nexists\langle$GARBLE, $sid$, $ssid\rangle$ :
        ignore this message
    record $\langle sid, ssid, (F, K, d)\rangle$
    // simulator requesting OT labels
    for $i = 1, \ldots, n$ :
        send (OT-RECEIVE, $(ssid, i)$) to $\mathcal{A}$
        record $\langle (ssid, i), \perp\rangle$

On a query $p$ to $H_1(\cdot)$

if $\exists\langle H_1, p, h\rangle$ :
    return $h$
else
    $h \overset{\$}{\leftarrow} \{0,1\}^n$
    record $\langle H_1, p, h\rangle$
    return $h$
```

Figure 3.6.: The Simulator Sim Part II. Simulation of Protocol Messages and the First Random Oracle H_1.

```
On $(\text{OT-Send}, (ssid, i), (X_i[0], X_i[1]))$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{OT}}$ on behalf of $\hat{\mathrm{S}}$
55 : record $\langle \hat{\mathrm{S}}, (ssid, i), (X_i[0], X_i[1]) \rangle$
56 : send $(\text{OT-Send}, (ssid, i))$ to $\mathcal{A}$.
57 : ignore further $(\text{OT-Send}, (ssid, i), \dots)$ messages

On $(\text{OT-Sent}, (ssid, i))$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{OT}}$
58 : if $\nexists \langle \hat{\mathrm{S}}, (ssid, i), (X_i[0], X_i[1]) \rangle$ :
59 :   ignore this message
60 : else
61 :   send $(\text{OT-Sent}, (ssid, i))$ to $\hat{\mathrm{S}}$
62 : ignore further $(\text{OT-Sent}, (ssid, i))$ messages

On $(\text{OT-Receive}, (ssid, i), x_i)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{OT}}$ on behalf of $\hat{\mathrm{U}}$
63 : record $\langle (ssid, i), x_i \rangle$
64 : send $(\text{OT-Receive}, (ssid, i))$ to $\mathcal{A}$
65 : ignore further $(\text{OT-Receive}, (ssid, i))$ messages

On $(\text{OT-Received}, (ssid, i))$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{OT}}$
66 : if $\nexists \langle \mathrm{S}, (ssid, i), (X_i[0], X_i[1]) \rangle$ or $\nexists \langle (ssid, i), x_i \rangle$ :
67 :   ignore this message
68 : elseif $x_i \neq \perp$
69 :   send $(\text{OT-Received}, (ssid, i), X_i[x_i])$ to $\hat{\mathrm{U}}$
70 : else
71 :   if $\forall r \in \{1, \dots, n\} \setminus \{i\} \exists \langle \text{OT-Received}, ssid, r \rangle$
72 :       and $(\exists \langle sid, ssid, (F, K, d) \rangle$
73 :       or $\exists \langle \mathrm{S}, sid, ssid, (F, K, d), X[0^n], X[1^n] \rangle)$ :
74 :     send $(\text{RcvCmplt}, sid, ssid, \mathrm{U}, \mathrm{S})$ to $\mathcal{F}_{\mathrm{OPRF}}$
75 :   else
76 :     record $\langle \text{OT-Received}, ssid, i \rangle$
```

Figure 3.7.: The Simulator Sim Part III. Simulation of $\mathcal{F}_{\mathrm{OT}}$.

```
On a new query $(p, y)$ to $H_2(\cdot,\cdot)$

if $\exists\langle H_2, p, y, \rho\rangle$ :
  return $\rho$
else
  if $\nexists\langle H_1, p, h = H_1(p)\rangle$ :
    $\rho \xleftarrow{\$} \{0,1\}^l$ and record $\langle H_2, p, y, \rho\rangle$
    return $\rho$
  else
    // check all simulated honest server S:
    if $\exists\langle \mathrm{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$, s.t. $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \,\|\, K)) = y$ :
      // $\mathsf{De}(d, \mathsf{Ev}(F, X[h] \,\|\, K))$ means $C_k(h)$ for the garbled $k$
      choose a new $ssid'$
      send $(\textsc{Eval}, sid, ssid', \mathrm{S}, p)$ to $\mathcal{F}_{\mathrm{OPRF}}$
      send $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathrm{S})$ to $\mathcal{F}_{\mathrm{OPRF}}$
      if $\mathcal{F}_{\mathrm{OPRF}}$ does not answer :
        output fail and abort
      else
        receive $(\textsc{EvalOut}, sid, ssid', \rho)$ from $\mathcal{F}_{\mathrm{OPRF}}$
        record $\langle H_2, p, y, \rho\rangle$
        return $\rho$
    else
      // check all corrupt server $\hat{S}$ with key $k_{\hat{S}}$ :
      if $\nexists\langle k_{\hat{S}}, \hat{S}\rangle$ s.t. $C_{k_{\hat{S}}}(h) = y$ :
        $\rho \xleftarrow{\$} \{0,1\}^l$ and record $\langle H_2, p, y, \rho\rangle$
        return $\rho$
      elseif there are multiple $k_{\hat{S}}$ : $C_{k_{\hat{S}}}(h) = y$ :
        output fail and abort
      else
        retrieve $\langle k_{\hat{S}}, \hat{S}\rangle$
        send $(\textsc{OfflineEval}, sid, \hat{S}, p)$ to $\mathcal{F}_{\mathrm{OPRF}}$
        receive $(\textsc{OfflineEval}, sid, \rho)$ from $\mathcal{F}_{\mathrm{OPRF}}$
        record $\langle H_2, p, y, \rho\rangle$
        return $\rho$
```

Figure 3.8.: The Simulator Sim Part IV. Simulation of the Second Random Oracle H_2.

4. Verifiability

An OPRF is said to have *verifiability* if the user can – roughly speaking – be sure that a server does not switch keys between several OPRF evaluations with the user. Thus, the outputs that the user received are all sampled from a fixed PRF $F_k(\cdot)$ where k is the fixed key of the server. To make this notion even useful, we must relax our requirements on the passively secure parties. If all parties always follow the protocol, the same server will always choose the same key. Therefore, we have that every passively secure OPRF is also a VOPRF.

We will assume in this section that corrupted servers may decide to choose a new key $k' \in \{0,1\}^m$ at will. By this, we consider strictly stronger adversaries as in Section 3.5. We still require that the adversaries behave honestly in garbling the circuit. That means we assume that every circuit F that is sent by a corrupted server to a user is calculated as $(F, e, d) \leftarrow \mathrm{Gb}(1^\lambda, \mathcal{VC})$, where $\mathcal{VC}$ is the circuit of the protocol description.

4.1. Adapting the Construction

In this section, we introduce the ideal functionality $\mathcal{F}_{\mathrm{VOPRF}}$ that captures the above security requirement rigorously. $\mathcal{F}_{\mathrm{VOPRF}}$ is depicted in Figure 4.1. The main difference to the ideal functionality $\mathcal{F}_{\mathrm{OPRF}}$ in Figure 3.1 is the message $(\textsc{Param}, \mathsf{S}, \pi)$ from the adversary $\mathcal{A}$ to $\mathcal{F}_{\mathrm{VOPRF}}$. The adversary $\mathcal{A}$ can send this message for a server identity S to set the identificator of that server. An identificator is some information that is published by the server as "fingerprint" of its key. A client will use this identificator to specify from which server it queries output. That means for the ideal functionality that $\mathcal{F}_{\mathrm{VOPRF}}$ keeps a table params of all server identities and their associated identificators. Note that the adversary is even allowed to choose the identificator for an honest server. If the adversary later allows the delivery of an output value to a user by sending $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, \pi)$ to $\mathcal{F}_{\mathrm{VOPRF}}$, the adversary has to specify the identificator π of the server from whose table $T_{sid}(\mathsf{S}, \cdot)$ the output should be taken. In other words, instead of specifying a server id i like in $\mathcal{F}_{\mathrm{OPRF}}$ of Figure 3.1, the adversary $\mathcal{A}$ specifies π to receive an output from a certain server. The mechanism for offline evaluation is adapted accordingly such that an indentificator has to be specified to receive the output of an offline evaluation.

Albrecht et al. [Alb+21] sketch an idea to construct a verifiable OPRF with garbled circuits. Their idea can be directly applied to our construction:

Let $H_3 : \{0,1\}^* \to \{0,1\}^\lambda$ be a third hash function. In an initialization phase, the server draws an uniformly random value $r \in \{0,1\}^\lambda$ and publishes the "fingerprint" $h_k = H_3(k \parallel r)$.

Now the definition of the protocol is changed in that the circuit jointly calculated by both parties will no longer be just a PRF, but will be the following function:

Functionality $\mathcal{F}_{\text{VOPRF}}$

For each value i and each session sid, an empty table $T_{sid}(i, \cdot)$ is initially undefined. Whenever $T_{sid}(i, x)$ is referenced below while it is undefined, draw $T_{sid}(i, x) \xleftarrow{\$} \{0, 1\}^l$.

Initialization:

- On $(\textsc{Init}, sid)$ from S, if this is the first Init message for sid, set $\mathsf{tx}(\mathsf{S}) = 0$ and send $(\textsc{Init}, sid, \mathsf{S})$ to $\mathcal{A}$. From now on, use "S" to denote the unique entity which sent the Init message for sid. Ignore all subsequent Init messages for sid.

- On $(\textsc{Param}, \mathsf{S}, \pi)$ from $\mathcal{A}$ if $\mathsf{params}[\mathsf{S}]$ is undefined then set $\mathsf{params}[\mathsf{S}] = \pi$.

Offline Evaluation:

On $(\textsc{OfflineEval}, sid, c, p)$ from $\mathsf{P} \in \{\mathsf{S}, \mathcal{A}\}$, send $(\textsc{OfflineEval}, sid, T_{sid}(i, p))$ to P if there is no entry $\mathsf{params}[i] = c$ and $\mathsf{P} = \mathcal{A}$ or if there is an entry $\mathsf{params}[i] = c$ and any of the following hold: (i) S is corrupted, (ii) P = S.

Online Evaluation:

- On $(\textsc{Eval}, sid, ssid, \mathsf{S}, p)$ from $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$, record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$ and send $(\textsc{Eval}, sid, ssid, \mathsf{P}, \mathsf{S})$ to $\mathcal{A}$.

- On $(\textsc{SndrComplete}, sid, ssid)$ from S, increment $\mathsf{tx}(\mathsf{S})$, send $(\textsc{SndrComplete}, sid, ssid, \mathsf{S})$ to $\mathcal{A}$.

- On $(\textsc{RcvCmplt}, sid, ssid, \mathsf{P}, \pi)$ from $\mathcal{A}$, retrieve $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$ (where $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$). Ignore this message if one of the following holds:
  - There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$.
  - If there exists an honest server S such that $\mathsf{params}[\mathsf{S}] = \pi$, but $\mathsf{tx}(\mathsf{S}) = 0$.
  - S' is honest but $\pi \neq \mathsf{S}'$.

  Send $(\textsc{Eval}, sid, T_{sid}(i, p))$ to P. If $\mathsf{params}[\mathsf{S}] = \pi$ decrement $\mathsf{tx}(\mathsf{S})$.

Figure 4.1.: The Ideal Functionality $\mathcal{F}_{\text{VOPRF}}$ Inspired by [BKW20; JKX18].

```
$\mathcal{VC}((h_k, x), (k, r))$
$y \coloneqq C_k(x)$
$b \coloneqq (h_k \overset{?}{=} H_3(k, r))$
$v \coloneqq ((1 - b) \cdot \bot) + (b \cdot y)$
```

C_k is still a boolean circuit that calculates a permutation F that is a PRF as defined in Definition 1. The server now has to provide its secret key k and its random value r to the garbled circuit and the client has to provide its input and the fingerprint h_k of the server from which he wants to retrieve the result. The circuit does not only compute the output of the PRF but does also check if the key has the claimed fingerprint. Only if that is true, the PRF result is output.

We can express this idea even a bit more generalized, by saying that the server calculates a commitment $(c, r) \leftarrow \mathsf{Commit}(k)$ as an "fingerprint" or identificator, where r is the opening information of the commitment c. So the circuit that will be garbled is the following:

```
$\mathcal{VC}((c, x), (k, r))$
$y \coloneqq C_k(x)$
$b \coloneqq \mathsf{Unveil}(c, k, r)$
$v \coloneqq ((1 - b) \cdot \bot) + (b \cdot y)$
```

If the commitment c can be opened to the value k using the decommitment information r, the PRF output is returned. Else, an error symbol $\bot$ is returned.

Applying this idea to Figure 3.2, we get the protocol depicted in Figures 4.2 to 4.3. Unusually for the UC-framework, we do *not* work in the $\mathcal{F}_{\mathrm{Com}}$-hybrid model. This is because we need the fact that we can express the unveil algorithm Unveil as a boolean circuit. We require that $\mathsf{COM} = (\mathsf{Commit}, \mathsf{Unveil})$ is computationally hiding and computationally binding. For the sake of simplicity, we assume that Commit outputs values in $\{0, 1\}^\lambda$. We denote the labels for the input (k, r) as KR and the labels for the input (c, x) as CX. There are only three major differences in this construction to the GC-OPRF construction from Figure 3.2. The first is of course, that the server now garbles the boolean circuit $\mathcal{VC}$ above. The second is that the server creates a commitment c when it receives an initialization message. The third is the hash function $H_2(\cdot, \cdot, \cdot)$. The user also hashes the commitment of the server by sending a (p, y, c) to H_2.

4.2. Proving Verifiability

In huge parts, the simulator for this proof works analogously to the simulator in Figures 3.5 to 3.8 for proving that GC-OPRF in Figure 3.2 UC-emulates $\mathcal{F}_{\mathrm{OPRF}}$. Therefore we will only elaborate on the differences. We depicted the routines with the essential differences in Figure 4.4. To make it easier for the reader to spot the differences between Figure 4.4 and the simulator from Figures 3.5 to 3.8, we marked all the lines that contain essential changes between the two simulators with a gray background.

```
S on $(\textsc{Init}, sid)$ from $\mathcal{E}$
If this is the first $(\textsc{Init}, sid)$ message from $\mathcal{E}$
$k \xleftarrow{\$} \{0,1\}^m$
$(c, r) \xleftarrow{\$} \mathsf{Commit}(k)$
record $\langle \mathsf{S}, c, r, k \rangle$
// Send identificator to U via $\mathcal{F}_{\text{AUTH}}$
send $(\textsc{Send}, mid, \mathsf{U}, (\textsc{Init}, c))$ to $\mathcal{F}_{\text{AUTH}}$

U on $(\textsc{Eval}, sid, ssid, \mathsf{S}, p)$ from $\mathcal{E}$
$x \xleftarrow{\$} H_1(p)$
send $(\textsc{Send}, mid, \mathsf{S}, (\textsc{Garble}, sid, ssid))$ to $\mathcal{F}_{\text{AUTH}}$

S on $(\textsc{SndrComplete}, sid, ssid)$ from $\mathcal{E}$
if already received $(\textsc{Garble}, sid, ssid)$:
  goto GarbleCircuit
else
  ignore this message

S on $(\textsc{Sent}, mid, \mathsf{U}, \mathsf{S}, (\textsc{Garble}, sid, ssid))$ from $\mathcal{F}_{\text{AUTH}}$
if already received $(\textsc{SndrComplete}, sid, ssid)$:
  GarbleCircuit:
  if $\nexists \langle \mathsf{S}, c, r, k \rangle$:
    ignore this message
  $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{VC})$
  $(CX[0] \,\|\, KR) := \mathsf{En}(e, 0^{\lambda+n} \,\|\, k \,\|\, r)$
  $(CX[1] \,\|\, KR) := \mathsf{En}(e, 1^{\lambda+n} \,\|\, k \,\|\, r)$
  send $(\textsc{Send}, mid', \mathsf{U}, (sid, ssid, (F, KR, d))$ to $\mathcal{F}_{\text{AUTH}}$
  for $i \in \{1, \ldots, \lambda + n\}$ :
    send $(\text{OT-}\textsc{Send}, (ssid, i), (CX_i[0], CX_i[1]))$ to $\mathcal{F}_{\text{OT}}$
else
  ignore this message
```

Figure 4.2.: Our Verifiable VGC-OPRF Construction Part I.

```
U on $(\textsc{Sent}, mid, \mathrm{S}, \mathrm{U}, (sid, ssid, (F, KR, d))$ from $\mathcal{F}_{\mathrm{AUTH}}$

if already received $(\textsc{Eval}, sid, ssid, \mathrm{S}, pw)$:
  wait for $(\text{OT-}\textsc{Sent}, (ssid, 1)), \ldots, (\text{OT-}\textsc{Sent}, (ssid, \lambda + n))$ from $\mathcal{F}_{\mathrm{OT}}$
  for $i \in \{1, \ldots, \lambda\}$ :
    send $(\text{OT-}\textsc{Receive}, (ssid, i), h_i)$ to $\mathcal{F}_{\mathrm{OT}}$
  for $i \in \{\lambda + 1, \ldots, \lambda + n\}$ :
    send $(\text{OT-}\textsc{Receive}, (ssid, i), x_i)$ to $\mathcal{F}_{\mathrm{OT}}$
  else
    ignore this message

U on $\{(\text{OT-}\textsc{Received}, (ssid, i), CX_i)\}_{i=1,\ldots,\lambda+n}$ from $\mathcal{F}_{\mathrm{OT}}$

if already received $(sid, ssid, (F, K, d))$:
    $Y \coloneqq \mathsf{Ev}(F, CX \,\|\, KR)$
    $y \coloneqq \mathsf{De}(d, Y)$
    if $y = \perp$:
      abort
    $\rho \xleftarrow{\$} H_2(pw, y)$
    output $(\textsc{Eval}, sid, ssid, \rho)$ to $\mathcal{E}$
else
  ignore this message
```

Figure 4.3.: Our Verifiable VGC-OPRF Construction Part II.

Not depicted are obvious changes, as e.g. that the simulator now has to garble the adapted circuit $\mathcal{VC}$ and that input labels are also created for the inputs c and r.

The first major difference is that Sim now has to react differently on $(\textsc{Init}, sid, \mathsf{S})$ messages from $\mathcal{F}_{\text{VOPRF}}$. These messages are sent to Sim, when a new honest server is initialized by the environment. In that case, Sim draws a uniformly random key k and commits to this key. The ideal functionality allows the adversary to choose the identificator c of a server, so Sim records the key and the commitment-randomness corresponding to this server and sends $(\textsc{Param}, \mathsf{S}, c)$ to $\mathcal{F}_{\text{VOPRF}}$. Finally, this c is also output as the output of the honest server S. Sim keeps records $\langle \mathsf{hon}, c, k, r\rangle$ for all identificators of honest servers.

A second difference is the reaction on an $(\textsc{Init}, c)$ messages from $\mathcal{A}$ on behalf of some corrupted server $\hat{\mathsf{S}}$. In this case, Sim just forwards the adversary's choice of an identificator c to $\mathcal{F}_{\text{VOPRF}}$ and records this identificator c. Sim keeps records $\langle \mathsf{corr}, c, \hat{\mathsf{S}}\rangle$ for all identificators of corrupted servers.

We also changed the response of the simulator to receiving a complete set of input labels via $(\text{OT-}\textsc{Received}, (ssid, 1)), \dots, (\text{OT-}\textsc{Received}, (ssid, \lambda + n))$. As before, if the requests of the labels were just simulated by Sim, i.e., $\langle (ssid, i) \bot\rangle$ for all $i \in \{1, \dots, \lambda + n\}$, it means that Sim must produce an output for the honest user. The simulator now uses the additional power of the garbled circuit that allows Sim to check if the encoded key K and the encoded opening information R can be opened to the commitment c. The simulator Sim If the garbled circuit F does not output $\bot$, the simulator Sim request output from $\mathcal{F}_{\text{VOPRF}}$ via $(\textsc{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S}, c)$. Else, Sim makes U abort the execution.

Finally, we consider the changes in responses on H_2 queries. The most notable change is that c is now a third argument to the hash function. This allows Sim to send $\textsc{RcvCmplt}$ and $\textsc{OfflineEval}$ messages with c as identificator to $\mathcal{F}_{\text{VOPRF}}$. Sim keeps a list of honest and a list of corrupted servers, i.e., their identificators. If c is in the list of honestly initialized servers, Sim knows the corresponding key k and can validate $C_k(h) = y$. This can be seen in line 32 Figure 4.4. This case is analogous to the case Figure 3.8 line 85, where Sim finds the key of an honest server.

If c is in the list of corrupted servers, Sim does *not* know the corresponding key to c. Remember that we assume that $\mathcal{A}$ may choose different keys in this chapter. As the server is corrupted, Sim cannot safely call $(\textsc{RcvCmplt}, sid, ssid, \mathcal{A}, c)$ for this server as there might have been a corresponding $(\textsc{RcvCmplt}, sid, ssid, \mathcal{A}, c)$ message generated as a result of all labels being received via OT. But in that case, Sim can safely send a $(\textsc{OfflineEval}, sid, c, p)$ message, as the server is corrupted. This does not influence the ticket counter of the server.

The proof of indistinguishability between $\text{EXEC}_{\text{IDEAL}_{\mathcal{F}_{\text{VOPRF}}}, \mathsf{Sim}, \mathcal{E}}$ and $\text{EXEC}_{\text{VGC-OPRF}, \mathcal{A}, \mathcal{E}}$ now works in many parts analogously to the proof in Section 3.5. We will only elaborate on the important difference and argue, why the output of an honest user in the ideal world is indistinguishable from the output of an honest user in the real world. We start with the differences in the proof:

- The most important difference in the proof is that the commitment c now has to be taken into account. The intuition is the following. As the commitment scheme COM is computationally hiding, it is safe for the simulator to send $(\textsc{Init}, c)$ for a simulated k and a commitment $(c, r) \leftarrow \mathsf{Commit}(k)$ on that key to a potentially corrupted U. An adversary $\mathcal{A}$ that can calculate some information about the key k with $(c, d) \leftarrow \mathsf{Commit}(k)$ would break the computationally hiding property of COM.

- A similar statement to Lemma 1 can be proven by using the computationally hiding property of the commitment scheme. The main idea is that if $\mathcal{A}$ provokes that Sim sends $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, c)$ in line 34 of Figure 4.4, then the server with identificator c must be an honest server. If $\mathcal{A}$ is able to make Sim send $(\text{RcvCmplt}, sid, ssid, \mathrm{S}, c)$ in line 20 in Figure 4.4, we are in an OPRF execution with an honest user. By a similar reduction, we receive a statement like Lemma 3, which says that for an honest server and an honest user, $\mathcal{A}$ can at most with negligible probability query $H_2(p, y, c)$ such that c is a commitment on the key k of the honest server and $C_k(H_1(p)) = y$. Because if the server is honest, $\mathcal{A}$ can at most with negligible probability calculate y. The adversary $\mathcal{A}$ knows two pieces of information that depend on k. The first is c. But if this would help $\mathcal{A}$ in calculating y, we could construct an adversary against the hiding property of COM, see Definition 4. The second piece of information is the garbling (F, KR, d). If that would help $\mathcal{A}$, we could construct an adversary against the *privacy* of the garbling scheme, see Definition 11. As $\mathcal{A}$ does not have any information on k, the best chance to compute $y = C_k(H_1(p))$ is by guessing, as C is a PRF as defined in Definition 1.

Honest User Output As already discussed in Section 3.5, in the real world, ρ is calculated as $\rho = H_2(p, \mathsf{De}(d, \mathsf{Ev}(F, CX \| KR)), c)$, where (F, KR, d) was generated by the server and CX are the labels received via OT for $x = H_1(p)$ and the identificatior commitment c. In the ideal world, ρ is chosen uniformly at random by $\mathcal{F}_{\mathrm{VOPRF}}$ if a fresh $(\text{Eval}, sid, ssid, \mathrm{S}, p)$ message was sent. If an honest user with input p interacts with S, the functionality $\mathcal{F}_{\mathrm{VOPRF}}$ will send $\rho = T_{sid}(\mathrm{S}, p)$ as output for the honest user. The simulator must produce the same output ρ for $H_2(p, y, c)$ if $y = C_k(H_1(p))$ and $\mathsf{Unveil}(c, k, r) = 1$ holds for S's key k and opening information r. Therefore, we have to compare the output of H_2 with the outputs of $\mathcal{F}_{\mathrm{VOPRF}}$. We distinguish the following cases in simulation of H_2:

Case 1: There is no record $\langle H_1, p, h\rangle$ found: Sim only needs to program the random oracle, if p, y, and c do occur in a protocol execution. More precisely, if $y = C_k(H_1(p))$ holds for some key k, where $\mathsf{Unveil}(c, k, r) = 1$ holds for some opening information r. That is, because in this case $\mathcal{F}_{\mathrm{VOPRF}}$ can eventually output a value ρ as the output of an honest user with input p and identificator c interacting with a server with key k and opening information r. We will call a query (p, y, c) *relevant* if there is a key k and an opening information r, such that $y = C_k(H_1(p))$ $\mathsf{Unveil}(c, k, r) = 1$. In the following, we bound the probability for the event that (p, y, c) becomes relevant, when $H_1(p)$ is not determined yet.

All keys $k_1, \ldots, k_t$ of honest servers are chosen independently. However, this time we also have to consider maliciously chosen keys from corrupted servers. The adversary $\mathcal{A}$ could choose keys $\hat{k}_1, \ldots, \hat{k}_s$ that are somehow correlated. However, that does not affect the following statement: Let $t \in \mathbb{N}$ be the number of servers in the protocol execution. Let $k_1, \ldots, k_t$ be the uniformly random and independently drawn keys

```
On (INIT, S, $sid$) from $\mathcal{F}_{\mathrm{VOPRF}}$

If this is the first (INIT, S, $sid$) message from $\mathcal{F}_{\mathrm{VOPRF}}$
    $k \xleftarrow{\$} \{0,1\}^m$
    $(c, r) \leftarrow \mathsf{Commit}(k)$
    record $\langle \mathsf{hon}, c, k, r \rangle$
    send (PARAM, S, $c$) to $\mathcal{F}_{\mathrm{VOPRF}}$
    send (INIT, $c$) as message from S to U

On (SEND, $mid$, U, (INIT, $c$)) from $\mathcal{A}$ on behalf of $\hat{S}$

send (SEND, $mid$, $\hat{S}$, U, (INIT, $c$)) to $\mathcal{A}$
on (OK, $mid$) from $\mathcal{A}$:
    record $\langle \mathsf{corr}, c, \hat{S} \rangle$
    send (PARAM, $\hat{S}$, $c$) to $\mathcal{F}_{\mathrm{VOPRF}}$

On (OT-RECEIVED, ($ssid$, $i$)) from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{OT}}$

if $\nexists \langle (ssid, i), (CX_i[0], CX_i[1]) \rangle$ or $\nexists \langle (ssid, i), x_i \rangle$:
    ignore this message
elseif $x_i \neq \bot$
    send (OT-RECEIVED, ($ssid$, $i$), $CX[x_i]$) to $\hat{U}$
else
    if $\forall t \in \{1, \ldots, n\} \setminus \{i\} \exists \langle$OT-RECEIVED$, ssid, t \rangle$:
            and $(\exists \langle sid, ssid, (F, KR, d) \rangle$:
            or $\exists \langle S, sid, ssid, (F, KR, d), CX[0^n], CX[1^n] \rangle)$:
        if $\mathsf{De}(d, \mathsf{Ev}(F, C[c] \,\|\, X[0] \,\|\, KR)) \neq \bot$:
            send (RCVCMPLT, $sid$, $ssid$, S, $c$) to $\mathcal{F}_{\mathrm{VOPRF}}$
        else
            ignore this message
    else
        record $\langle$OT-RECEIVED$, ssid, i \rangle$

On a new query $(p, y, c)$ to $H_2(\cdot, \cdot, \cdot)$

if $\exists \langle H_2, p, y, c, \rho \rangle$:
    return $\rho$
else
    if $\nexists \langle H_1, p, h = H_1(p) \rangle$:
        $\rho \xleftarrow{\$} \{0,1\}^l$ and record $\langle H_2, p, y, c, \rho \rangle$
        return $\rho$
    else
        if $\exists \langle \mathsf{hon}, c, k, r \rangle$ and $C_k(h) = y$:
            send (EVAL, $sid$, $ssid'$, $p$) to $\mathcal{F}_{\mathrm{VOPRF}}$
            send (RCVCMPLT, $sid$, $ssid'$, $\mathcal{A}$, $c$) to $\mathcal{F}_{\mathrm{VOPRF}}$
            if $\mathcal{F}_{\mathrm{VOPRF}}$ does not answer:
                output fail and abort
            else
                receive (EVAL, $sid$, $ssid'$, $\rho$) from $\mathcal{F}_{\mathrm{VOPRF}}$
                record $\langle H_2, p, y, c, \rho \rangle$
                return $\rho$
        elseif $\exists \langle \mathsf{corr}, c, \hat{S} \rangle$
            send (OFFLINEEVAL, $sid$, $c$, $p$) to $\mathcal{F}_{\mathrm{VOPRF}}$
            receive (OFFLINEEVAL, $sid$, $\rho$) from $\mathcal{F}_{\mathrm{VOPRF}}$
            record $\langle H_2, p, y, c, \rho \rangle$
            return $\rho$
        else
            $\rho \xleftarrow{\$} \{0,1\}^l$ and record $\langle H_2, p, y, c, \rho \rangle$
            return $\rho$
```

Figure 4.4.: The Major Changes to Get a Simulator Sim for $\mathcal{F}_{\mathrm{VOPRF}}$.

used by the – honest or corrupted– servers. Let C be the PRF function calculated by $\mathcal{VC}$ and let $n \in \Omega(\lambda)$ be the output length of C. We assumed in the beginning that $C_{k_i}(\cdot)$ is a permutation. Thus, if we choose some uniformly random input $x \in \{0,1\}^n$, we get that $C_{k_i}(x) \in \{0,1\}^n$ is uniformly random. If $H_1(p)$ is not queried yet, we have for every $i \in \{1,\dots,t\}$ and every $y \in \{0,1\}^n$:

$$\Pr[C_{k_i}(H_1(p)) = y] \leq \frac{1}{2^n},$$

where the probability is taken over the random output of H_1. Thus, we get by a union-bound that the probability for a key to make (p, y, c) *relevant* is at most $t2^{-n}$, which is negligible.

Case 2: Records $\langle H_1, p, h\rangle$ and $\langle \mathsf{hon}, c, k, r\rangle$ exist, such that $C_k(h) = y$:

In this case, the value h is the output of the random oracle H_1 on input p. As the commitment scheme COM is correct, we have that $\mathsf{Unveil}(c, k, r) = 1$, because Sim calculated c and r as $(c, r) \leftarrow \mathsf{Commit}(k)$, see Definition 3. The tuple (p, y, c) is relevant, because the key of an honest server produces the output y, when the input h is provided to the circuit, and the circuit does not output $\perp$, because $\mathsf{Unveil}(c, k, r) = 1$. Thus, Sim programs $H_2(p, y, c)$. The simulator Sim sends $(\textsc{Eval}, sid, ssid', \mathsf{S}, p)$ to $\mathcal{F}_{\mathrm{VOPRF}}$ for a new subsession id $ssid'$. That means, Sim initiates a new protocol execution and requests itself the output value $\rho = T_{sid}(\mathsf{S}, p)$ from $\mathcal{F}_{\mathrm{VOPRF}}$. We argued in Section 4.2 why a similar statement to Lemma 1 holds. Thus, Sim can safely send the $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, c)$ message, without decreasing the ticket counter of S to 0. The random oracle $H_2(p, y, c)$ is programmed to the answer ρ of $\mathcal{F}_{\mathrm{VOPRF}}$. The programming ensures that $\mathcal{E}$ will get the same output $\rho = H_2(p, y, c)$ when invoking an execution of the protocol between an honest user with input p and the honest server with identificator c.

Case 3: There are records $\langle H_1, p, h\rangle$ and $\langle \mathsf{corr}, c, \hat{\mathsf{S}}\rangle$:

In that case, the value h is the output of the random oracle H_1 on input p, but c is the identificator of a corrupted server. The simulator Sim sends $(\textsc{OfflineEval}, sid, \hat{S}, p)$ to $\mathcal{F}_{\mathrm{VOPRF}}$ and receives the answer $(\textsc{OfflineEval}, sid, \rho)$ from $\mathcal{F}_{\mathrm{VOPRF}}$. Sim programs $H_2(p, y, c)$ to the output ρ of the offline evaluation. Sim does *not* check if the key k on which c is a commitment does even output y. In fact, Sim does not even know that key, as $\mathcal{A}$ just sent $(\textsc{Param}, \hat{\mathsf{S}}, c)$ to Sim. But Sim knows that sending $(\textsc{OfflineEval}, sid, c, p)$ will not affect the ticket counter. And if for some key k and some opening information r with $\mathsf{Unveil}(c, k, r) = 1$ it would hold that $y \neq y' = C_k(h)$, it would mean that the tuple (p, y, c) is not relevant, i.e., there will be no real-world execution of the protocol, where an honest user would request (p, y, c). An honest user would query the $H_2(p, y', c)$ instead. In that case, Sim has unnecessarily programmed H_2. But as the programming was done with a uniformly random value, the output of H_2 is still indistinguishable from a uniformly random value.

Contrarily, if c is actually a commitment on a key k such that $\mathcal{A}$ knows opening information r such that $\mathsf{Unveil}(c, k, r) = 1$ and $y = C_k(h)$, the oracle $H_2(p, y, c)$ is programmed to the right ρ. That is because the commitment scheme is computationally binding. That means, $\mathcal{A}$ can at most with negligible probability find another key k' and opening information r' such that $\mathsf{Unveil}(c, k', r') = 1$. That means for an OPRF execution between an honest user with input p and a corrupted server, the server can send a garbling (F, KR', d) at most with negligible probability such that $\perp \neq \mathsf{De}(d, \mathsf{Ev}(F, CX \,\|\, KR'))$, where CX are the labels for c and $h = H_1(p)$ and KR' are the labels for k' and r'.

Case 4: If c is never "registered" as identificator via a $(\textsc{Param}, \mathsf{S}, c)$ message, $H_2(p, y, c)$ is set to a uniformly random value. In this case, no user received a $(\textsc{Init}, c)$ message. Thus, no honest real-world user will input this c to the random oracle H_2.

5. Comparison of Concrete Efficiency

As we were interested in the concrete efficiency of our construction, we implemented it and compared it to other OPRF protocols. For the implementation, we leveraged a C++ framework, called EMP-Toolkit [WMK16]. Further, we implemented a version of the state-of-the-art OPRF protocol, 2HashDH, by [JKK14; Jar+16; JKX18]. Finally, we also compared the two former protocols to the lattice-based protocol of Albrecht et al. [Alb+21]. This protocol was already implemented by [Alb+21]. The main goal was to compare the concrete efficiency of different OPRFs on the same computer. All source code described below can be found in the GitHub repository `github.com/SebastianFaller/OPRF-Garbled-Circuits`. The benchmark results refer to the version of commit `ad35dbf01dc8bf4f09f2bd839aa36bda042675e6`. Update to latest commit before deadline.

5.1. Garbled-Circuit-Based OPRF

We will introduce our implementation in two steps. First, we present the implementation of a generic garbling scheme reminiscent of the formal definition from Section 2.6.3. The source code for the garbling scheme was written in collaboration with the supervisors of the thesis.

5.1.1. Implementing the Garbling Scheme

The EMP-Toolkit is a framework that offers various routines for the efficient calculation of garbled circuits and other cryptographic building blocks, such as hashing and symmetric encryption. To the best of our knowledge, almost all relevant garbled circuit optimizations, including Free-XOR [KS08] and Half-Gates [ZRE15], are implemented. Only the newly published Three-Halves technique from Rosulek and Roy [RR21] is not yet implemented. As an example of usage, we showed how garbling of a certain circuit can be realized in C++ using EMP-Toolkit.

EMP-Toolkit processes circuits that are described in *Bristol Format* [Arc+]. Bristol Format is a specification of how to encode algebraic or boolean circuits. EMP-Toolkit already has a description of AES in Bristol Format built-in.

EMP-Toolkit allows to load a circuit from a Bristol Format file via the class `BristolFormat`. This can be done by using the constructor of the class. The statement `BristolFormat cf( circuit_filename.c_str());` constructs a Bristol Format circuit object named `cf` when given a path to the file as string `circuit_filename`. We'd like to emphasize that this is not the garbled circuit yet, but rather a description of the plain boolean circuit. EMP-Toolkit defines the type `HalfGateGen<T>`. By creating an object of this type, the programmer determines:

- With which optimizations the circuit will be calculated. `HalfGateGen` is the class that implements “Half-Gates” but there is e.g. a class `PrivacyFreeGen` that implements a garbling scheme that is even more efficient but has no privacy.

- Where is the garbled circuit written to. This is done via the type parameter `T`. EMP-Toolkit has several input-output classes that can be specified as a type parameter. For instance, if `T` is `NetIO`, the garbled circuit is directly sent over the network. If `FileIO` is chosen, the circuit is written to a file on the machine. We like to note here that solving this problem via C++ Templates might not be an optimal choice, as it is elusive to programmers which types might be used as type parameters without thoroughly knowing the framework. Further, it disallows dynamic changing of the desired behavior at runtime. A better solution would have been the “strategy” design pattern [Gam10].

To generate a garbling we also have to assign random labels to all input bits. The basic unit of computation in EMP-Toolkit is the type `block`. By default, all garbling routines offer 128 bits of security, so a `block` has 128 bits. That means each input label will be one block of pseudo-random data. We generate these blocks by using EMP-Toolkit’s PRG: `prg.random_block(input, n);` fills the `block`- array `input` with n pseudo-random blocks of data. The same has to be done for the output labels. Afterwards the circuit can be garbled by using `cf.compute(output, input_1, input_2);`, where `input_1`, `input_2`, and `output` are the above calculated arrays.

The listing in Listing 5.1 shows the whole code for garbling a circuit. Note that the listing also shows how further values as the encoding information and decoding information are computed.

```cpp
void garble(IOType* io, vector<block>* encoding_info, vector<bool>* decoding_info, const
 string& circuit_filename) {
HalfGateGen<IOType>::circ_exec = new HalfGateGen<IOType>(io);
BristolFormat cf(circuit_filename.c_str());
encoding_info->resize(cf.n1+cf.n2+1);
decoding_info->resize(cf.n3);
block* input_1 = new block[cf.n1];
block* input_2 = new block[cf.n2];
block* output = new block[cf.n3];
PRG prg;
prg.random_block(input_1, cf.n1);
prg.random_block(input_2, cf.n2);
//garble the circuit
cf.compute(output, input_1, input_2);
//write decoding info
for(int i=0; i<cf.n3; i++) {
    (*decoding_info)[i] = getLSB(output[i]);
}
//write encoding info
(*encoding_info)[0] = ((HalfGateGen<IOType>*) HalfGateGen<IOType>::circ_exec)->delta;
for (int i=0; i<cf.n1; i++) {
    (*encoding_info)[i+1] = input_1[i];
}

```

```

    for (int i=0; i<cf.n2; i++) {
        (*encoding_info)[cf.n1+1+i] = input_2[i];
    }
    //Clean up
    delete HalfGateGen<IOType>::circ_exec;
    delete[] input_1;
    delete[] input_2;
    delete[] output;
    }

```

Listing 5.1: Garbling a Circuit Using EMP-Toolkit

Similar to the above listing, one can implement a whole garbling interface, inspired by the formal definition from Section 2.6.3. We get the interface in Listing 5.2. The function `void garble(...)` is as described above. This function corresponds to the function $(F, e, d) = \mathsf{Gb}(1^\lambda, f)$ of Section 2.6.3.

The function `void encode(...)` takes a vector `input` of boolean values as argument. This vector contains all input bits that shall be encoded. The vector `encoding_info` contains the encoding information that was output by `void garble(...)`. The final labels will be stored in the vector `encoded_input`. This function corresponds to the function $X = \mathsf{En}(e, x)$ of Section 2.6.3.

The function `void evaluate(...)` takes an object for handling input and output as the first argument. The garbled circuit itself – meaning F in terms of Section 2.6.3 – will be read from this object. The function also takes some input labels `encoded_input` and a string `circuit_filename` that points to the Bristol format description of the circuit as argument. The circuit will be evaluated on the specified input labels and will be stored in the vector `encoded_output`. This function corresponds to the function $Y = \mathsf{Ev}(F, X)$ of Section 2.6.3.

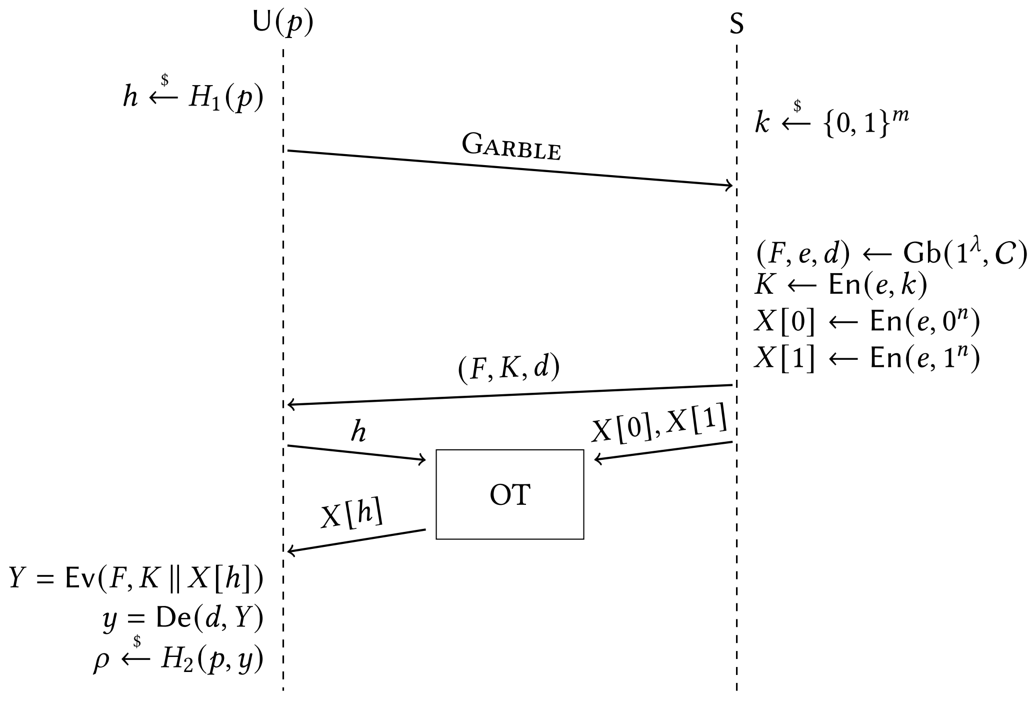
The function `void decode(...)` takes the `encoded_output` that was calculated by `void evaluate (...)` and the `decoding_info` calculated by `void garble(...)` and stores the final output as vector of boolean values in `output`. This function corresponds to the function $y = \mathsf{De}(d, Y)$ of Section 2.6.3.

```
template <class IOType>
void garble(IOType* io, vector<block>* encoding_info, vector<bool>* decoding_info, const
  string& circuit_filename)
}
void encode(vector<emp::block>* encoded_input, const vector<bool>& input, const vector<
emp::block>& encoding_info);

template <class IOType>
void evaluate(IOType* io, vector<block>* encoded_output, const vector<block>&
encoded_input, const string& circuit_filename);
}

void decode(vector<bool>* output, const vector<emp::block>& encoded_output, const vector
<bool>& decoding_info);


```

Listing 5.2: Garbling Scheme Interace

5.1.2. Implementing the Protocol Parties

We employed the above scheme to implement the protocol parties for GC-OPRF. We used AES as concrete instantiation for the circuit C in our implementation. An overview of the protocol flow can be found in Figure 5.1.



U(p)
S
$h \xleftarrow{\$} H_1(p)$
$k \xleftarrow{\$} \{0,1\}^m$
GARBLE
$(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$
$K \leftarrow \mathsf{En}(e, k)$
$X[0] \leftarrow \mathsf{En}(e, 0^n)$
$X[1] \leftarrow \mathsf{En}(e, 1^n)$
(F, K, d)
h
$X[0], X[1]$
OT
$X[h]$
$Y = \mathsf{Ev}(F, K \parallel X[h])$
$y = \mathsf{De}(d, Y)$
$\rho \xleftarrow{\$} H_2(p, y)$

Figure 5.1.: Overview of GC-OPRF.

We modeled the protocol by creating a class for the user and a class for the server. The user has four member functions that allow interaction with the server. The function `bool* eval(string pwd, int ssid)` hashes the password of the user using SHA3 and returns the input bits for the circuit as `bool*`. The function `void receiveLabels(bool* choices, block* encoded_user_input)` uses EMP-Tool's OT interface to exchange the labels for the user input. For the sake of simplicity, we did not implement a provably UC-secure OT protocol but retreated to use the already implemented Naor-Pinkas OT [NP01] from EMP-Toolkit. We describe the protocol for the interested reader in Appendix A.3. With `void receiveKeyAndDecoding(block* encoded_key, bool* decoding_info)`, the user receives the labels for the key and the decoding information via OT. Note that in contrast to the actual protocol description in Figure 3.2, the garbling of the circuit F is sent *after* the key labels and the decoding information K, d. This is because EMP-Tool consumes the garbled circuit directly from the network interface when the circuit is evaluated. Finally, `uint8_t* onLabelsReceived(int ssid, const block* encoded_user_input, const block* encoded_key, const bool* decoding_info)` evaluates the circuit, decodes the output and hashes the output with SHA3. This is depicted in Listing 5.3. Note that certain constants as `ip_addr` and `AES_KEY_SIZE` are defined elsewhere. We will not go into the details of each called function, as they are simply using the garbled circuit scheme, described above, and the hash function SHA3.

```
NetIO user_io(ip_addr, port); // User is the OT-Receiver.
User<NetIO> u(sid, &user_io);

bool* current_h = u.eval(password, ssid);

// Receive garbled encoded key and decoding info
block encoded_key[AES_KEY_SIZE];
bool decoding_info[AES_INPUT_SIZE];
u.receiveKeyAndDecoding(encoded_key, decoding_info);

// Request labels via OT for H_1(password)
block labels[AES_INPUT_SIZE];
u.receiveLabels(current_h, labels);

// Evalute the circuit and hash the output
uint8_t* output = u.onLabelsReceived(ssid, labels, encoded_key, decoding_info);

```

Listing 5.3: User Execution of GC-OPRF

To create a server, one needs to choose a uniformly random key. The most important method of the server is `void onGarble(int ssid, vector<block>* encoded_ones, vector<block>* encoded_zeroes)`. This function creates a garbled circuit, decoding information, and input labels. Note that we first use the EMP class `MemIO` to garble the circuit. This is again because the garbling routines produce direct output to the network interface when the circuit is garbled. Letting this output go to local memory instead of the network interface facilitates the sending of the remaining data.

```
void onGarble(int ssid, vector<block>* encoded_ones, vector<block>* encoded_zeroes){
//Write garbled circuit to memory first, so other data is sent first
MemIO* mem_io = new MemIO();
vector<block> encoding_info;
vector<bool> decoding_info;
garble(mem_io, &encoding_info, &decoding_info, circuit_filename);

vector<bool> input_zeros(AES_INPUT_SIZE, false);
//append key
input_zeros.insert(input_zeros.end(), key.begin(), key.end());
encode(encoded_zeroes, input_zeros, encoding_info);
vector<block> encoded_key = vector<block>(encoded_zeroes->begin() + AES_INPUT_SIZE,
 encoded_zeroes->begin() + (AES_INPUT_SIZE+AES_KEY_SIZE));


encoded_zeroes->resize(AES_INPUT_SIZE);

vector<bool> input_ones(AES_INPUT_SIZE, true);
encode(encoded_ones, input_ones, encoding_info);

// Send everything to the user
sendKeyAndDecoding(encoded_key, decoding_info);

sendLabelsOverOT(*encoded_zeroes, *encoded_ones);

sendGarbledCircuitFromMem(mem_io);
```

```
    }

```

Listing 5.4: Server's response to a Garble message

5.2. The 2HashDH Protocol

For the implementation of the 2HashDH protocol from [Jar+16; JKK14] we relied on the OPENSSL library [OPENSSL] in version 1.1.1. OPENSSL is a commercial-grade open-source library for cryptography and secure communication and is already installed on most Linux systems. In particular, we used the algorithms for elliptic curve cryptography to instantiate the 2HashDH protocol. The protocol is depicted on a high level in Figure 5.2. Note that we will use additive group notation in this chapter.

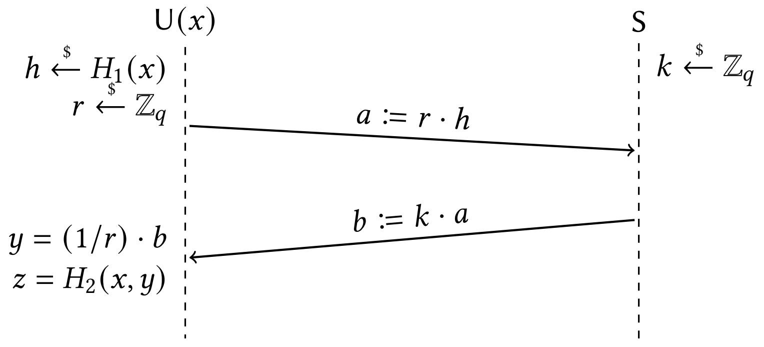


Figure 5.2.: Overview of 2HashDH.

A slightly simplified listing of the code for the 2HashDH user can be seen in Listing 5.5. Note that we still used EMP-Toolkit for the network communication. This dependency could easily be removed to make our implementation more portable.

The user starts by initializing a group. We decided to use the NIST P256 curve [Sta19], as it offers 128 bits of security, which is comparable to the garbled circuit implementation of EMP-Toolkit. The group is initialized by `EC_GROUP_new_by_curve_name(NID_X9_62_prime256v1)`. Usually in OPENSSL, one also needs to specify a pointer to the `BN_CTX` structure, which can be described as a buffer for certain calculations. Next, by calling `EC_GROUP_get_curve()`, one gets the parameters of the elliptic curve. In particular, we are interested in the order of the underlying field, as we will need this value later to invert the blinding value `r`. The first important step of the protocol is to hash the input string `pwd` to a point on the elliptic curve. Note that this is by far the most involved part of the protocol. We will elaborate on it in Appendix A.1. After receiving a point `g` of the password, the user chooses a random blinding value by calling `BN_rand_range(r, field_order)`. With `EC_POINT_mul(ec_group, a, NULL , g, r, bn_ctx)`, the product $r \cdot g$ is calculated, using additive group notation. The resulting point is then sent to the server. The server will execute similar code to multiply the received point with its key and send it back. Now the user calculates the inverse of $r \in \mathbb{F}$ with `BN_mod_inverse(oneOverR, r, field_order, bn_ctx)` and multiplies this with the received point from the server by calling `EC_POINT_mul(ec_group, y, NULL, b, oneOverR, bn_ctx)`. Finally, the resulting point is hashed using SHA3.

```
// Create group object for NIST P256 curve
EC_GROUP* ec_group = EC_GROUP_new_by_curve_name(NID_X9_62_prime256v1);
BN_CTX* bn_ctx = BN_CTX_new();
EC_GROUP_precompute_mult(ec_group, bn_ctx);
// 32 byte for field element and one for encoding byte
const int ec_point_size_comp = 33;

// order needed to create and invert r
BIGNUM* field_order = BN_new();
EC_GROUP_get_curve(ec_group, field_order, NULL, NULL, bn_ctx);

EC_POINT* g = hash_to_curve(pwd, ec_group, bn_ctx);

BIGNUM* r = BN_new();
//Choose random r
BN_rand_range(r, field_order);
EC_POINT* a = EC_POINT_new(ec_group);
// a = g*r
EC_POINT_mul(ec_group, a, NULL, g, r, bn_ctx);

uint8_t buf[ec_point_size_comp];
// Convert point to raw binary data
EC_POINT_point2oct(ec_group, a, POINT_CONVERSION_COMPRESSED, buf, ec_point_size_comp,
 bn_ctx);

user_io.send_data(buf, ec_point_size_comp);

// Receive b from server
EC_POINT* b = EC_POINT_new(ec_group);
user_io.recv_data(buf, ec_point_size_comp);
BIGNUM* oneOverR = BN_new();
BN_mod_inverse(oneOverR, r, field_order, bn_ctx);
EC_POINT* y = EC_POINT_new(ec_group);
// y = (1/r)*b
EC_POINT_mul(ec_group, y, NULL, b, oneOverR, bn_ctx);

// Hash the resulting point
EC_POINT_point2oct(ec_group, y, POINT_CONVERSION_COMPRESSED, buf, ec_point_size_comp,
 bn_ctx);
uint8_t hashTwo[32]; // 32 bytes sha3 output

sha3_256(hashTwo, buf, ec_point_size_comp);

```

Listing 5.5: User for 2HashDH

5.3. Lattice-based OPRF

To get a plausibly post-quantum secure OPRF-protocol as another comparison to our construction, we chose the lattice-based OPRF from Albrecht et al. [Alb+21]. In their work, Albrecht et al. [Alb+21] implemented a proof of concept of their protocol in SageMath

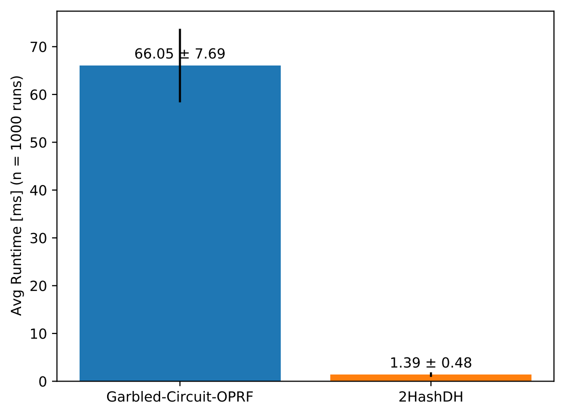
Protocol	**Avg. Runtime** [ms]	**Network Traffic** [kB]
Our work	66.05 ± 7.69	241.541
2HashDH [JKX18]	1.39 ± 0.48	0.066
Albrecht et al. [Alb+21]	7406.008 ± 54.890	513.254 ± 0.170

Figure 5.3.: Overview of the Benchmark Results

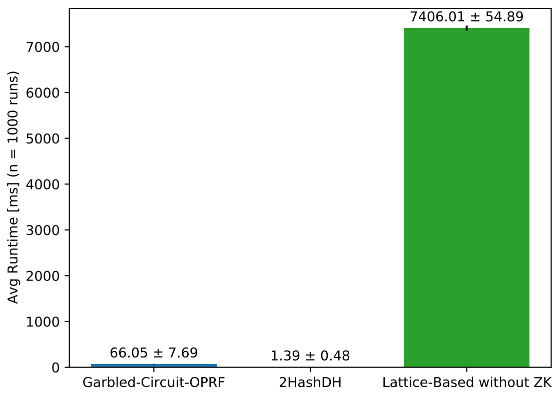
[Ste05]. For the sake of simplicity, all zero-knowledge proofs were left out in the implementation. We benchmarked this SageMath implementation. However, the comparison to our protocol can only be seen as a rough estimate of actual efficiency. On the one hand, an implementation of the protocol in C++ as we wrote for our construction and 2HashDH would lead to significantly better performance. That is because SageMath is an interpreted language, based on Python, while C++ is compiled. On the other hand, the performance impact of zero-knowledge proofs in a lattice-based setting can be enormous. Albrecht et al. [Alb+21, Sec 5.3] estimate that using the state-of-the-art lattice-based zero-knowledge proof from [Yan+19] would result in more than 2^{40} bits of communication. Another point that makes this comparison less reliable is the estimation of lattice parameters. In general, it is considered a non-trivial task to choose appropriate lattice parameters in order to achieve a required security level. We opted to choose similar parameters as for the National Institute of Standards and Technology (NIST) post-quantum competition algorithm NewHope [Alk+16, Protocol 3] as both claim a security level of 128 bits, which is the same security level we have for the 2HashDH implementation and the implementation of our construction. Namely, we let the lattice dimension $n = 1024$, chose the prime modulus q as a 14 bit prime and let the rounding modulus $p = 3$. Note, that this parameter choice is likely over-optimistic and should not be considered for real-world implementations of the protocol. Albrecht et al. [Alb+21] estimate the parameters for their scheme themselves and their estimations are far more pessimistic. They suggest $n = 16384$, a prime modulus q with around 256 bit and a rounding modulus that is polynomial in λ. We tried these parameters for our benchmarks but SageMath would abort the execution of even a single protocol instance i.e., one exchanged PRF value, with a failure message. We believe that the test laptop for our benchmarks does not have enough memory. Therefore, we retreated to the smaller parameters, mentioned above.

5.4. Benchmarks

We tested the three implementations on an Intel Core i5-5200U CPU @ 2.20GHz × 4 on the local network interface. We measured the running time in milliseconds that each program needs from the invocation of an OPRF session until the user calculated the output. The server used the same PRF key for all executions. We also measured the amount of data that the protocols exchange over the network, meaning data sent from user to server and vice-versa. We summarized the results in Figure 5.3.



(a) Running times of GC-OPRF and 2HashDH [JKX18].



(b) Running Times of GC-OPRF, 2HashDH [JKX18], and [Alb+21].

Figure 5.4.: Comparison of the Measured Running Times.

5.4.1. Running Time

We depicted the results for the running time measurement in Figure 5.4. We measured an average running time of 66.05 ms for our own GC-OPRF protocol, with a standard deviation of 7.69 ms. We measured an average running time of 1.39 ms for 2HashDH, with a standard deviation of 0.48 ms. With under two milliseconds, the 2HashDH protocol by [JKX18] was about 50 times faster than our construction. This is not surprising as the protocol merely needs to exchange two points of an elliptic curve. We found a noticeable difference in running time to the lattice-based construction of [Alb+21]. We measured an average running time of 7559.70 ms for the [Alb+21] protocol, with a standard deviation of 184.61 ms. Our construction is over 110 times faster than the lattice-based protocol. We like to note here that the difference might get slightly smaller when the communication goes over a high-latency Wide Area Network (WAN). This is because the protocol from [Alb+21] requires only two rounds of communication, while our construction requires four rounds.

5.4.2. Network Traffic

We depicted the results for the network traffic measurement in Figure 5.5. Our construction sends 241.541 kB of data over the network. 2HashDH by [JKX18] sends only two points on the NIST P256 curve, which is exactly 66 B. We measured about 513.474 kB of network traffic with a standard deviation of 96 kB for the lattice-based protocol by [Alb+21]. Note that the network traffic of our GC-OPRF implementation and of 2HashDH are constant values, while there are slight variations in the measurement for the protocol of [Alb+21]. This is because the transmitted value in the protocol is a random element in a cyclotomic ring modulo some prime number. SageMath automatically compresses those elements if possible which leads to a varying size.

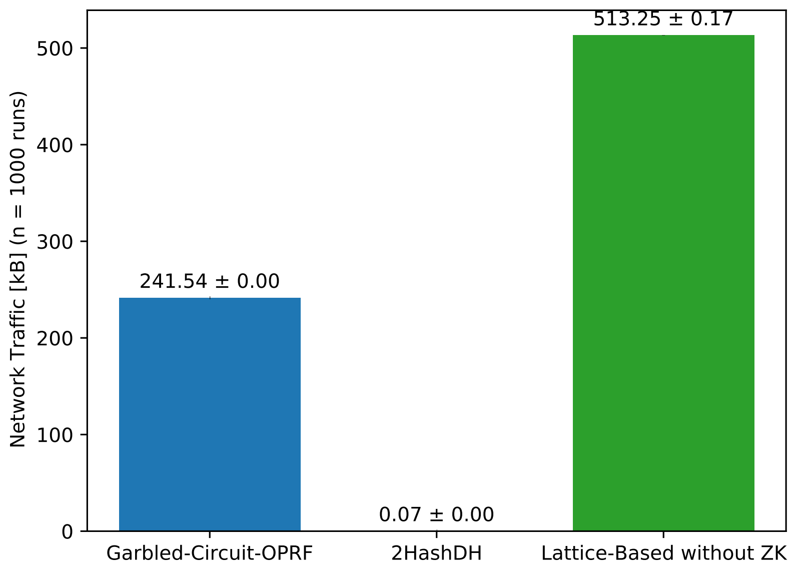


Figure 5.5.: Comparison of the Measured Network Traffic.

The measured network traffic for our GC-OPRF implementation matches our theoretical estimates. We estimated around 230 kB of traffic for our construction as follows: According to [Arc+], the employed AES circuit has 6400 and-gates. Each and-gate requires two ciphertexts to be transmitted. The used ciphertext in EMP-Toolkit is 128 bit long. Thus, we have 32 B of data of each and-gate. This makes $6400 \cdot 32\,\text{B} = 204\,800\,\text{B}$. Additionally, we have 128 executions of a Naor-Pinkas OT. This OT protocol is DLOG-based and EMP-Toolkit implements a variant with elliptic curves. EMP-Toolkit uses the same NIST P-256 elliptic curve for OT as we did for 2HashDH. However, they do represent a group element uncompressed as 65 B of data. We reduced this cost to 33 B in our 2HashDH implementation by using a compressed representation from OPENSSL [OPENSSL]. One does not need to store an x- and a y-coordinate for a point on an elliptic curve. It is sufficient to store the x-coordinate and the sign. A single 1-out-of-2 OT requires the transfer of three group elements and two ciphertexts (see our description of Naor-Pinkas OT in Appendix A.3). Again, a ciphertext is 128 bit, i.e., 16 B long. All OT executions sum up to $128(3 \cdot 65\,\text{B} + 2 \cdot 16\,\text{B}) = 29\,056\,\text{B}$. In total, we have $204\,800\,\text{B} + 29\,056\,\text{B} = 233\,856\,\text{B}$. We assume that the difference to the actually measured value comes from meta-data and other overhead produced by EMP-Toolkit.

6. Conclusion

In this work, we investigated the security of a garbled-circuit-based OPRF in the UC-framework [Can01]. To realize an ideal OPRF functionality in the style of Jarecki, Krawczyk, and Xu [JKX18], we augmented the "straightforward" construction of Pinkas et al. [Pin+09] with a second hash function. This second hash function was modeled as random oracle and allowed the simulator in the proof to "program" the output of the random oracle to the values that the ideal functionality outputs. The resulting protocol is secure against passive adversaries.

We further used a technique proposed by Albrecht et al. [Alb+21] to make our OPRF verifiable. We changed to garbled circuit such that the user now provides a commitment on the key of the server. The server provides the key and the opening information to the circuit. Only if the commitment correctly opens to the key of the server, the garbled circuit outputs a pseudo-random value.

We implemented a prototype of our protocol and the state-of-the-art OPRF protocol 2HashDH by [Jar+16; JKK14; JKX18]. We compared the two implementations to a simplified implementation of the lattice-based OPRF by Albrecht et al. [Alb+21]. The experiments showed that our construction is significantly faster than the lattice-based protocol. We also found that our construction is not as efficient as the DLOG-based 2HashDH protocol. Nonetheless, the efficiency is still in a reasonable range with a running time of around 65 ms and around 250 kB network traffic. This indicates, that circuit-based OPRF protocols might be a promising candidate for post-quantum secure OPRFs.

Future Work Our security proof holds only for passive, i.e., honest-but-curious adversaries. This is a common assumption in cryptography, but it does arguably not capture realistic scenarios. Hence, a proof considering active adversaries is desirable.

We also expect that there is space for improvement concerning the choice of the circuit that is calculated by the garbling scheme. In our work, we assumed the circuit to be a PRF. But then the actual PRF that is calculated by the OPRF protocol is $f_k(x) = H_2(p, \mathsf{F}_k(H_1(p)))$, where F is a PRF. One could suspect that modeling H_2 as a random oracle already introduces enough entropy to the output of the function. But we leave it for future work if weaker assumptions on the circuit are sufficient to still achieve a secure protocol.

Additionally, we believe that more experimental insight would be beneficial. It would be good to also take network latency into account for the experiments. This means concretely, that the protocols should also be tested over a Local Area Network (LAN) and a WAN. It would also be interesting to compare our protocols with the batched, related-key OPRF of Kolesnikov et al. [Kol+16], as this protocol is also circuit-based and relies on OT.

Besides, we argued that circuit-based OPRF are promising candidates for post-quantum secure OPRFs. To strengthen this claim, it would be desirable to also implement our protocol using a – presumably – post-quantum secure OT protocol, e.g., [PVW08]. This would show if the "price" for post-quantum security is still in a reasonable range. Such a construction would need to be proven secure in the QROM model.

Finally, it would be interesting to see if our construction's necessity to program the random oracle is inherent to UC-secure OPRFs. Hesse [Hes20] showed that aPAKE cannot be achieved without a programmable random oracle. As Jarecki, Krawczyk, and Xu [JKX18] carved out the close connection between aPAKE and OPRF, it is an intriguing question if one can by connecting both works show that UC-secure OPRFs require a programmable random oracle.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. First Edition. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4.

[Alb+21] Martin R. Albrecht et al. “Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices”. In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, May 2021, pp. 261–289. DOI: 10.1007/978-3-030-75248-4_10.

[Alk+16] Erdem Alkim et al. “Post-quantum Key Exchange - A New Hope”. In: *USENIX Security 2016: 25th USENIX Security Symposium*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 327–343.

[Amy+16] Matthew Amy et al. “Estimating the Cost of Generic Quantum Pre-image Attacks on SHA-2 and SHA-3”. In: *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. St. John’s, NL, Canada: Springer, Heidelberg, Germany, Aug. 2016, pp. 317–337. DOI: 10.1007/978-3-319-69453-5_18.

[Arc+] David Archer et al. *’Bristol Fashion’ MPC Circuits*. URL: https://homes.esat.kuleuven.be/~nsmart/MPC/ (visited on 02/04/2022).

[Aru+19] Frank Arute et al. “Quantum Supremacy Using a Programmable Superconducting Processor”. In: *Nature* 574.7779 (7779 Oct. 2019), pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5.

[Bas+21] Andrea Basso et al. “Cryptanalysis of an Oblivious PRF from Supersingular Isogenies”. In: *Advances in Cryptology – ASIACRYPT 2021*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 160–184. ISBN: 978-3-030-92062-3. DOI: 10.1007/978-3-030-92062-3_6.

[Bau+16] Bela Bauer et al. “Hybrid Quantum-Classical Approach to Correlated Materials”. In: *Physical Review X* 6.3 (Sept. 21, 2016), p. 031045. DOI: 10.1103/PhysRevX.6.031045.

[Bel+08] Mira Belenkiy et al. *Delegatable Anonymous Credentials*. Cryptology ePrint Archive, Report 2008/428. https://eprint.iacr.org/2008/428. 2008.

[Ben+11] Rikke Bendlin et al. "Semi-homomorphic Encryption and Multiparty Computation". In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 169–188. DOI: 10.1007/978-3-642-20465-4_11.

[BG90] Mihir Bellare and Shafi Goldwasser. "New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero Knowledge Proofs". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 194–211. DOI: 10.1007/0-387-34805-0_19.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of garbled circuits". In: *ACM CCS 2012: 19th Conference on Computer and Communications Security*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. Raleigh, NC, USA: ACM Press, Oct. 2012, pp. 784–796. DOI: 10.1145/2382196.2382279.

[BKW20] Dan Boneh, Dmitry Kogan, and Katharine Woo. "Oblivious Pseudorandom Functions from Isogenies". In: *Advances in Cryptology – ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 2020, pp. 520–550. DOI: 10.1007/978-3-030-64834-3_18.

[Blu+91] Manuel Blum et al. "Checking the Correctness of Memories". In: *32nd Annual Symposium on Foundations of Computer Science*. San Juan, Puerto Rico: IEEE Computer Society Press, Oct. 1991, pp. 90–99. DOI: 10.1109/SFCS.1991.185352.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols (Extended Abstract)". In: *22nd Annual ACM Symposium on Theory of Computing*. Baltimore, MD, USA: ACM Press, May 1990, pp. 503–513. DOI: 10.1145/100216.100287.

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Transactions on Symmetric Cryptology* 2019.2 (2019), pp. 55–93. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i2.55-93.

[Bon+11] Dan Boneh et al. "Random Oracles in a Quantum World". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Seoul, South Korea: Springer, Heidelberg, Germany, Dec. 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3.

[Bri+10] Eric Brier et al. "Efficient Indifferentiable Hashing into Ordinary Elliptic Curves". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA:

Springer, Heidelberg, Germany, Aug. 2010, pp. 237–254. DOI: 10.1007/978-3-642-14623-7_13.

[BS20] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Jan. 2020. URL: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf.

[Büs+20] Niklas Büscher et al. “Secure Two-Party Computation in a Quantum World”. In: *ACNS 20: 18th International Conference on Applied Cryptography and Network Security, Part I*. Ed. by Mauro Conti et al. Vol. 12146. Lecture Notes in Computer Science. Rome, Italy: Springer, Heidelberg, Germany, Oct. 2020, pp. 461–480. DOI: 10.1007/978-3-030-57808-4_23.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. “Constrained Key-Homomorphic PRFs from Standard Lattice Assumptions - Or: How to Secretly Embed a Circuit in Your PRF”. In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, Mar. 2015, pp. 1–30. DOI: 10.1007/978-3-662-46497-7_1.

[Can+02] Ran Canetti et al. “Universally composable two-party and multi-party secure computation”. In: *34th Annual ACM Symposium on Theory of Computing*. Montréal, Québec, Canada: ACM Press, May 2002, pp. 494–503. DOI: 10.1145/509907.509980.

[Can00] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. https://eprint.iacr.org/2000/067. 2000.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *42nd Annual Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[Can98] Ran Canetti. *Security and Composition of Multi-party Cryptographic Protocols*. Cryptology ePrint Archive, Report 1998/018. https://eprint.iacr.org/1998/018. 1998.

[CF01] Ran Canetti and Marc Fischlin. “Universally Composable Commitments”. In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 19–40. DOI: 10.1007/3-540-44647-8_2.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. “Tracing Traitors”. In: *Advances in Cryptology – CRYPTO’94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 257–270. DOI: 10.1007/3-540-48658-5_25.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. “The Random Oracle Methodology, Revisited (Preliminary Version)”. In: *30th Annual ACM Symposium on Theory of Computing*. Dallas, TX, USA: ACM Press, May 1998, pp. 209–218. DOI: 10.1145/276698.276741.

[Cha83] David Chaum. “Blind Signature System”. In: *Advances in Cryptology – CRYPTO’83.* Ed. by David Chaum. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1983, p. 153.

[Cho+13] Seung Geol Choi et al. “Efficient, Adaptively Secure, and Composable Oblivious Transfer with a Single, Global CRS”. In: *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography.* Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Nara, Japan: Springer, Heidelberg, Germany, Feb. 2013, pp. 73–88. DOI: `10.1007/978-3-642-36362-7_6`.

[Dav+18] Alex Davidson et al. “Privacy Pass: Bypassing Internet Challenges Anonymously”. In: *Proceedings on Privacy Enhancing Technologies* 2018.3 (July 2018), pp. 164–180. DOI: `10.1515/popets-2018-0026`.

[Dav+22] Alex Davidson et al. *Oblivious Pseudorandom Functions (OPRFs) Using Prime-Order Groups.* Internet-draft draft-irtf-cfrg-voprf-09. Internet Engineering Task Force / Internet Engineering Task Force, Feb. 8, 2022. 63 pp. URL: `https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-09`.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Berlin Heidelberg: Springer-Verlag, 2002. ISBN: 978-3-540-42580-9. DOI: `10.1007/978-3-662-04722-4`.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. “A Verifiable Random Function with Short Proofs and Keys”. In: *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography.* Ed. by Serge Vaudenay. Vol. 3386. Lecture Notes in Computer Science. Les Diablerets, Switzerland: Springer, Heidelberg, Germany, Jan. 2005, pp. 416–431. DOI: `10.1007/978-3-540-30580-4_28`.

[Faz+20] Faz-Hernandez et al. *Internet Draft: Hashing to Elliptic Curves.* Apr. 27, 2020. URL: `https://tools.ietf.org/id/draft-irtf-cfrg-hash-to-curve-07.html` (visited on 02/04/2022).

[Fre+05] Michael J. Freedman et al. “Keyword Search and Oblivious Pseudorandom Functions”. In: *TCC 2005: 2nd Theory of Cryptography Conference.* Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 303–324. DOI: `10.1007/978-3-540-30576-7_17`.

[Gam10] Erich Gamma, ed. *Design Patterns: Elements of Reusable Object-Oriented Software.* 38. printing. Addison-Wesley Professional Computing Series. Boston, Mass.: Addison-Wesley, 2010. XV, 395 S. : Ill., graph. Darst. ISBN: 978-0-201-63361-0.

[GK90] Oded Goldreich and Hugo Krawczyk. “On the Composition of Zero-Knowledge Proof Systems”. In: *Automata, Languages and Programming.* Ed. by Michael S. Paterson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1990, pp. 268–282. ISBN: 978-3-540-47159-2. DOI: `10.1007/BFb0032038`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[Hes20] Julia Hesse. “Separating Symmetric and Asymmetric Password-Authenticated Key Exchange”. In: *SCN 20: 12th International Conference on Security in Communication Networks*. Ed. by Clemente Galdi and Vladimir Kolesnikov. Vol. 12238. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2020, pp. 579–599. DOI: 10.1007/978-3-030-57990-6_29.

[Ish+03] Yuval Ishai et al. “Extending Oblivious Transfers Efficiently”. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[Jar+16] Stanislaw Jarecki et al. *Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online)*. Cryptology ePrint Archive, Report 2016/144. https://eprint.iacr.org/2016/144. 2016.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. “Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model”. In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 233–253. DOI: 10.1007/978-3-662-45608-8_13.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. “OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks”. In: *Advances in Cryptology – EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 2018, pp. 456–486. DOI: 10.1007/978-3-319-78372-7_15.

[JL09] Stanislaw Jarecki and Xiaomin Liu. “Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection”. In: *TCC 2009: 6th Theory of Cryptography Conference*. Ed. by Omer Reingold. Vol. 5444. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, Mar. 2009, pp. 577–594. DOI: 10.1007/978-3-642-00457-5_34.

[KBR13] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. “DupLESS: Server-Aided Encryption for Deduplicated Storage”. In: *USENIX Security 2013: 22nd USENIX Security Symposium*. Ed. by Samuel T. King. Washington, DC, USA: USENIX Association, Aug. 2013, pp. 179–194.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. "Improved OT Extension for Transferring Short Secrets". In: *Advances in Cryptology – CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 54–70. DOI: 10.1007/978-3-642-40084-1_4.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Second Edition. Vol. Chapman & Hall/CRC cryptography and network security. Boca Raton: CRC Press, 2015. ISBN: 978-1-4665-7027-6.

[Kol+16] Vladimir Kolesnikov et al. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl et al. Vienna, Austria: ACM Press, Oct. 2016, pp. 818–829. DOI: 10.1145/2976749.2978381.

[KS08] Vladimir Kolesnikov and Thomas Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". In: *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*. Ed. by Luca Aceto et al. Vol. 5126. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, Heidelberg, Germany, July 2008, pp. 486–498. DOI: 10.1007/978-3-540-70583-3_40.

[KsS12] Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. "Billion-Gate Secure Computation with Malicious Adversaries". In: *USENIX Security 2012: 21st USENIX Security Symposium*. Ed. by Tadayoshi Kohno. Bellevue, WA, USA: USENIX Association, Aug. 2012, pp. 285–300.

[LP07] Yehuda Lindell and Benny Pinkas. "An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries". In: *Advances in Cryptology – EUROCRYPT 2007*. Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Heidelberg, Germany, May 2007, pp. 52–78. DOI: 10.1007/978-3-540-72540-4_4.

[Mal+04] Dahlia Malkhi et al. "Fairplay - Secure Two-Party Computation System". In: *USENIX Security 2004: 13th USENIX Security Symposium*. Ed. by Matt Blaze. San Diego, CA, USA: USENIX Association, Aug. 2004, pp. 287–302.

[MF06] Payman Mohassel and Matthew Franklin. "Efficiency Tradeoffs for Malicious Two-Party Computation". In: *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Moti Yung et al. Vol. 3958. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 2006, pp. 458–473. DOI: 10.1007/11745853_30.

[Mos18] Michele Mosca. "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?" In: *IEEE Security Privacy* 16.5 (Sept. 2018), pp. 38–41. ISSN: 1558-4046. DOI: 10.1109/MSP.2018.3761723.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. “Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology”. In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 21–39. DOI: 10.1007/978-3-540-24638-1_2.

[Nie+12] Jesper Buus Nielsen et al. “A New Approach to Practical Active-Secure Two-Party Computation”. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 681–700. DOI: 10.1007/978-3-642-32009-5_40.

[NP01] Moni Naor and Benny Pinkas. “Efficient Oblivious Transfer Protocols”. In: *12th Annual ACM-SIAM Symposium on Discrete Algorithms*. Ed. by S. Rao Kosaraju. Washington, DC, USA: ACM-SIAM, Jan. 2001, pp. 448–457.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. “Privacy Preserving Auctions and Mechanism Design”. In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC ’99. New York, NY, USA: Association for Computing Machinery, Nov. 1, 1999, pp. 129–139. ISBN: 978-1-58113-176-5. DOI: 10.1145/336992.337028.

[NR04] Moni Naor and Omer Reingold. “Number-theoretic constructions of efficient pseudo-random functions”. In: *Journal of the ACM* 51.2 (2004), pp. 231–262.

[OPENSSL] *OPENSSL*. Copyright © 1999-2021 The OpenSSL Project Authors. All Rights Reserved. URL: https://www.openssl.org/ (visited on 02/04/2022).

[Pai99] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology – EUROCRYPT’99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[Pin+09] Benny Pinkas et al. “Secure Two-Party Computation Is Practical”. In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, Dec. 2009, pp. 250–267. DOI: 10.1007/978-3-642-10366-7_15.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. “A Framework for Efficient and Composable Oblivious Transfer”. In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2008, pp. 554–571. DOI: 10.1007/978-3-540-85174-5_31.

[Rab05] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Cryptology ePrint Archive, Report 2005/187. https://eprint.iacr.org/2005/187. 2005.

[RR21] Mike Rosulek and Lawrence Roy. “Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits”. In: *Advances in Cryptology – CRYPTO 2021, Part I.* Ed. by Tal Malkin and Chris Peikert. Vol. 12825. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, Aug. 2021, pp. 94–124. DOI: `10.1007/978-3-030-84242-0_5`.

[Sho94] Peter W. Shor. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”. In: *35th Annual Symposium on Foundations of Computer Science.* Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 1994, pp. 124–134. DOI: `10.1109/SFCS.1994.365700`.

[Sta19] National Institute of Standards and Technology. *Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters.* Draft Special Publication (SP) 800-186, Comments Due: January 29, 2020 (public comment period is CLOSED). Washington, D.C.: U.S. Department of Commerce, Oct. 2019. URL: `https://doi.org/10.6028/NIST.SP.800-186-draft`.

[Ste05] William Stein. *Sage Mathematical Software System.* Version 9.5 released 2022-01-30. 2005. URL: `https://www.sagemath.org/`.

[Ula07] Maciej Ulas. *Rational Points on Certain Hyperelliptic Curves over Finite Fields.* June 11, 2007. arXiv: `0706.1448 [math]`. URL: `http://arxiv.org/abs/0706.1448` (visited on 01/24/2022).

[WB19] Riad S. Wahby and Dan Boneh. “Fast and simple constant-time hashing to the BLS12-381 elliptic curve”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.4 (2019). `https://tches.iacr.org/index.php/TCHES/article/view/8348`, pp. 154–179. ISSN: 2569-2925. DOI: `10.13154/tches.v2019.i4.154-179`.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. *EMP-Toolkit: Efficient MultiParty Computation Toolkit.* emp-toolkit. 2016. URL: `https://github.com/emp-toolkit/emp-tool` (visited on 07/20/2021).

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation”. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security.* Ed. by Bhavani M. Thuraisingham et al. Dallas, TX, USA: ACM Press, Oct. 2017, pp. 21–37. DOI: `10.1145/3133956.3134053`.

[Yan+19] Rupeng Yang et al. “Efficient Lattice-Based Zero-Knowledge Arguments with Standard Soundness: Construction and Applications”. In: *Advances in Cryptology – CRYPTO 2019, Part I.* Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2019, pp. 147–175. DOI: `10.1007/978-3-030-26948-7_6`.

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *27th Annual Symposium on Foundations of Computer Science.* Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 1986, pp. 162–167. DOI: `10.1109/SFCS.1986.25`.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. “Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates”. In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 220–250. DOI: 10.1007/978-3-662-46803-6_8.

A. Appendix

In this chapter we present additional material for our thesis.

A.1. Implementing the Hash to Curve Algorithm

The 2HashDH construction described in Section 5.2 assumes the existence of a hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$, where $\mathbb{G}$ is the group in which the protocol operates and thus in which calculation of discrete logarithms is hard. In our case, it will be the group of points on the standardized elliptic curve NIST P-256 [Sta19]. The naive approach would be to hash the input string to a bit string of a predefined length $l \in \mathbb{N}$ and then to map the output bit string to a point on the elliptic curve. The group $\mathbb{G}$ has prime order and thus, the mapping $m : \{0,1\}^l \rightarrow \mathbb{G}$ cannot be bijective for $l > 1$. One could interpret the bit string as an integer and reduce it modulo the group order. Unfortunately, this is not sufficient. The proof of security for 2HashDH modeled H_1 as a random oracle. Thus, the output distribution must be "close to" uniformly random. But the described naive approach yields a skewed distribution.

Instead, a proper hash to curve algorithm must be employed. The Internet Engineering Task Force (IETF) proposed several algorithms in the internet draft [Faz+20]. We depict the algorithm recommended for the NIST P-256 curve in Figure A.1.

First, the input message *msg* is hashed using SHA256 to two field elements $u[0]$ and $u[1]$ of the underlying field of the elliptic curve. Each of these field elements is mapped to a point on the curve, using the `map_to_curve` algorithm. Then the two resulting points are added, using the addition of $\mathbb{G}$. In general, one must use a `clear_cofactor` algorithm to make sure that the resulting point lies in a subgroup of prime order. However, as the curve NIST P-256 already has prime order, this process can be left out in our case. We will describe the subroutines below.

```
hash_to_curve($msg$)
$u :=$ hash_to_field($msg, 2$)
$Q0 :=$ map_to_curve($u[0]$)
$Q1 :=$ map_to_curve($u[1]$)
$R := Q0 + Q1$
$P :=$ clear_cofactor($R$)
return $P$
```

Figure A.1.: Hash to Curve Algorithm.

```
SSWU($u, A, B, Z$)
$tv1 := \texttt{inv0}(Z^2 \cdot u^4 + Z \cdot u^2)$
$x1 := (-B/A) \cdot (1 + tv1)$
if $tv1 \overset{?}{=} 0$
    set $x1 := B/(Z \cdot A)$
$gx1 := x1^3 + A \cdot x1 + B$
$x2 := Z \cdot u^2 \cdot x1$
$gx2 := x2^3 + A \cdot x2 + B$
if $\texttt{is\_square}(gx1)$
    set $x := x1$ and $y := \texttt{sqrt}(gx1)$
else
    set $x := x2$ and $y := \texttt{sqrt}(gx2)$
if $\texttt{sgn0}(u) \neq \texttt{sgn0}(y)$
    set $y := -y$
return $(x, y)$
```

Figure A.2.: Simplified Shallue-van de Woestijne-Ulas Mapping.

Hash to Field The requirement on the hash to field algorithm is to be indifferentiable from a random oracle. This is a different notion than indistinguishable, see [MRH04]. The message is expanded to a sufficiently long string of bits by using several calls to SHA256. The resulting bit string is divided into smaller bit strings, one for each required field element. Next, the bit strings are interpreted as integers and reduced modulo the prime order of the field.

Map to Curve The internet draft [Faz+20] recommends to use the algorithm shown in Figure A.2. This algorithm maps a field element $u \in \mathbb{F}$ to a point $P = (x, y) \in \mathbb{G}$, where P is a point on a Weierstrass curve with equation

$$Y^2 = X^3 + AX + B,$$

with $A \neq 0, B \neq 0$. The algorithm is called *Simplified Shallue-van de Woestijne-Ulas* mapping. It was described by Brier et al. [Bri+10] and Ulas [Ula07] and enhanced by Wahby and Boneh [WB19]. The value $Z \in \mathbb{F}$ is a constant that depends on the curve. The function `inv0`(e) calculates the multiplicative inverse of $e \in \mathbb{F}$ or outputs 0 if $e = 0$. The function `is_square`(e) checks if e is a square in $\mathbb{F}$. If an element $e \in \mathbb{F}$ is square, the square root is calculated by the function `sqrt`(e). The function `sgn0`(e) returns 1 if e is positive or e is 0. Else it returns 0.

A.2. Advanced Encryption Standard

Advanced Encryption Standard is by far the most widely used block cipher. It was standardized by the NIST as successor of the Data Encryption Standard (DES). The algorithm is also called the Rijndael algorithm and was proposed by Daemen and Rijmen [DR02]. It is a block cipher and works on blocks of size 128 bits. Though Rijndael can work with different key-lengths, we will present the algorithm only for a key length of 256 bits.

The algorithm performs 14 rounds to encrypt one block of data. From a high point of view, the algorithm proceeds in the following order. We will explain each step in detail in the coming sections.

- Key expansion (generate round keys from original key)
- Add round key
- For round 1 to round 13
  - Sub bytes
  - Shift rows
  - Mix columns
  - Add round key
- Sub bytes
- Shift rows
- Add round key

A.2.1. Key expansion

The original key $k \in \{0,1\}^{256}$ is used to generate round keys for the 15 *Add Round Key* executions – one for each round plus the initial *Add RoundKey*. First, the original key is organized in words of 32 bits $W_0, \ldots, W_7$. These words are the first 8 round keys. The following round keys are defined recursively. For $i = 8, \ldots, 60$, we let:

$$W_i := \begin{cases} W_{i-8} \oplus S(W_{i-1} \lll 8) \oplus \texttt{const}(i), & \text{if } i \equiv 0 \bmod 8 \\ W_{i-8} \oplus S(W_{i-1}) \oplus \texttt{const}(i), & \text{if } i \equiv 4 \bmod 8 \\ W_{i-8} \oplus W_{i-1}, & \text{else} \end{cases}$$

The value of the constant $\texttt{const}(i)$ depends on i, by $\cdot \lll 8$ we denote rotating 8 bits to the left and S is a so-called S-box. This S-box substitutes the bytes of a word. We will explain it in Appendix A.2.3.

A.2.2. Add Round Key

In this step, each byte of the current state is combined via bitwise xor with the corresponding byte of the round key. Note that the round keys are of the same size as the states. This is the only step, where the key directly influences the result.

A.2.3. Sub Bytes

This step consist of replacing every byte by another byte, using the so-called S-box. This S-box describes a substitution and ensures that the algorithm is non-linear. First, the respective byte is interpreted as an element $x \in \mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$. If $x \neq 0$, replace x by $x' := x^{-1}$. Second, an affine transformation is applied to get the output $y = Ax + b$, for constants $A \in \{0,1\}^{8\times 8}, b \in \{0,1\}^8$, see [DR02] for the exact values of A and b.

A.2.4. Shift Rows

For this and the next step, the bytes of the current state are arranged in a 4×4 matrix. Then, the each row of the matrix is shifted by a certain offset. The first row is not shifted. The second row is shifted by one column to the left. The third row is shifted by two columns to the left and finally, the fourth row is shifted by three columns to the left.

A.2.5. Mix Columns

This step is performed in all rounds, except the final round. Again, we arrange the bytes of the current state as a 4×4 matrix. A column is now interpreted as a vector $(a_0, a_1, a_2, a_3)^\top \in \mathbb{F}_{2^8}^4$ and multiplied by a constant matrix as follows:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

A.3. Naor-Pinkas-OT

In this section we describe the OT protocol introduced by Naor and Pinkas [NP01]. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q for which the Computational Diffie-Hellman (CDH) assumption holds. Let $H : \mathbb{G} \longrightarrow \{0,1\}^\lambda$ be a random oracle. In the protocol, a sender S interacts with a receiver R. The sender gets as input two messages $M_0, M_1 \in \{0,1\}^\lambda$. The receiver gets as input $\sigma \in \{0,1\}$ and outputs $M_\sigma \in \{0,1\}^\lambda$. The protocol proceeds as follows:

- Initially, S chooses a random value $C \in \mathbb{G}$ (it is important that R does not know the discrete logarithm of C).
- R chooses $k \overset{\$}{\longleftarrow} \mathbb{Z}_q^*$ uniformly at random and sets $pk_\sigma := g^k$ and $pk_{1-\sigma} := C \cdot (pk_\sigma)^{-1}$. R sends pk_0 to S.
- S calculates $pk_1 = C \cdot (pk_1)^{-1}$. S chooses $r_0, r_1 \overset{\$}{\longleftarrow} \mathbb{Z}_q^*$ uniformly at random. S sets $E_0 = (g^{r_0}, H(pk_0^{r_0}) \oplus M_0)$ and $E_1 = (g^{r_1}, H(pk_1^{r_1}) \oplus M_1)$. S sends (E_0, E_1) to R.
- R computes $M_\sigma = H((E_\sigma[0])^k) \oplus E_\sigma[1]$, where E_σ denotes the first component of E_σ and $E_\sigma[1]$ denotes the second component.

Security. Intuitively, R's privacy comes from the fact that pk_0 is independent of σ. S's privacy comes from the fact that if R could calculate the discrete logarithm of both pk_0 and pk_1, R could also calculate the discrete logarithm of C. As H is a random oracle, an adversary that would decrypt both M_0 and M_1 would need to calculate $pk_0^{r_0}$ and $pk_1^{r_1}$. But as CDH holds in the group, an adversary has at most negligible advantage to calculate both of the two values, as $(g, g^{r_b}, pk_b, pk_b^{r_b})$ is a CDH-tuple if the adversary does not know the discrete logarithm of pk_b.

A.4. Actively Secure Garbled Circuits

Yao's garbled circuits as described above are only secure if the garbling party behaves honest-but-curious, i.e. passive adversaries. This is evident as a cheating garbler could just garble a different circuit than the operator expects. Think for instance about a circuit that outputs the evaluator's input. As the garbling scheme offers privacy, the evaluating party cannot know from $Y = \mathsf{Ev}(F, X)$ that the actually encoded output Y is its own input (without having the decoding information d).

Therefore, the evaluator must ensure that the garbled circuit he receives is indeed the circuit he expects, e.g., the one specified by the protocol.

A.4.1. Cut-and-Choose

The one of the first techniques in the literature that ensured the garbling of the right circuit was "cut-and-choose". This technique was used before in other contexts and was first applied to garbled circuits by Lindell and Pinkas [LP07]. The core idea is the following: The garbler does not only garble a single version garbling of the circuit but it creates many garblings of the same circuit, where each garbling is calculated with fresh randomness. When the circuits are sent to the evaluator the evaluator can demand from the operator to "open" certain gates and thus show, that the right gates was garbled. If the garbler fails to answer one of the evaluator's opening requests the evaluator aborts. If the garbler behaves honestly he can answer all requests of the evaluator. On the other hand, if the garbler altered the circuit there will be a negligibly small probability that the garbler can answer all request correctly.

However the big downside of this approach is the efficiency. To get statistical security in the security parameter λ, the garbler has to garble $\mathcal{O}(\lambda)$ circuits.

A.4.2. Authenticated Garbling

Wang, Ranellucci, and Katz [WRK17] introduced a method called *authenticated garbling* to ensure security of Yao's garbled circuits against malicious adversaries. The main idea is to use the information theoretic Message Authentication Code (MAC) from [Nie+12]. This MAC allows two parties A and B to authenticate a bit $b \in \{0, 1\}$. A holds a *global* key $\Delta_\mathsf{A} \in \{0, 1\}^\lambda$ which was chosen uniformly at random. This global key will be the same for all MACs generated by A. To authenticate a bit b held by B, A choses a *local* key

Functionality $\mathcal{F}_{\text{Pre}}$

- Upon receiving Δ_A from A and init from B, and assuming no values $\Delta_\mathsf{A}, \Delta_\mathsf{B}$ are currently stored, choose uniform $\Delta_\mathsf{B} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and store $\langle \Delta_\mathsf{A}, \Delta_\mathsf{B} \rangle$. Send Δ_B to B.
- Upon receiving $(\mathsf{random}, r, M[r], K[s])$ from A and random from B, sample uniform $s \in \{0,1\}$ and set $K[r] := M[r] \oplus r\Delta_\mathsf{B}$ and $M[s] := K[s] \oplus s\Delta_\mathsf{A}$. Send $(s, M[s], K[r])$ to B.
- Upon receiving $(\mathsf{AND}, (r_1, M[r_1], K[s_1]), (r_2, M[r_2], K[s_2]), (r_3, M[r_3], K[s_3]))$ from A and $(\mathsf{AND}, (s_1, M[s_1], K[r_1]), (s_2, M[s_2], K[r_2]))$ from B, verify that $M[r_i] = K[r_i] \oplus r_i\Delta_\mathsf{B}$ and that $M[s_i] = K[s_i] \oplus s_i\Delta_\mathsf{A}$, for $i \in \{1,2\}$. Send cheat to B if one of the checks fails. Otherwise, set $s_3 := r_3 \oplus ((r_1 \oplus s_1) \wedge (r_2 \oplus s_2))$, set $K[r_3] := M[r_3] \oplus r_3\Delta_\mathsf{B}$, and set $M[s_3] := K[s_3] \oplus s_3\Delta_\mathsf{A}$. Send $(s_3, M[s_3], K[r_3])$ to B.

Figure A.3.: The Ideal Functionality $\mathcal{F}_{\text{Pre}}$ From [WRK17].

$K[b] \in \{0,1\}^\lambda$ and we let the MAC be

$$M[b] := K[b] \oplus b\Delta_\mathsf{A}.$$

Let's ignore for a moment how the two parties calculate (or exchange) these values securely. A holds the local and the global key $(K[b], \Delta_\mathsf{A})$ and B holds the bit value b and the MAC $M[b]$. If B maliciously wanted to claim a different bit $b^* \neq b$, he would need to guess the local key $K[b]$, which is only possible with negligible probability as $K[b]$ is chosen uniformly at random for every new bit b. We will adher to the notation of [WRK17] and write $[b]_\mathsf{B}$ to denote the situation where B holds $(b, M[b] = K[b] \oplus b\Delta_\mathsf{A})$ and A holds $K[b]$ (and Δ_A). Symmetrically we write $[b]_A$ if A holds $(b, M[b] = K[b] \oplus b\Delta_\mathsf{B})$ and B holds a local key $K[b]$ and global key Δ_B.

Next, we note that the above scheme is XOR-homomorphic. Concretely, if e.g. A holds two authenticated bits $[b]_\mathsf{A}$ and $[c]_\mathsf{A}$ for $b, c \in \{0,1\}$, then A can locally compute $(b \oplus c, M[b \oplus c] = M[b] \oplus M[c])$ and B can locally compute $K[b \oplus c] = K[b] \oplus K[c]$ to get $[b \oplus c]_\mathsf{A}$. This XOR-homomorphism allows to combine the MAC with techniques from secret sharing.

Roughly speaking, these MACs will be used to authenticate the garbling of each gate of the garbled circuit.

Wang, Ranellucci, and Katz [WRK17] use the ideal functionality $\mathcal{F}_{\text{Pre}}$, depicted in Figure A.3 to realize their protocol. This ideal functionality "encapsulates" the preprocessing phase for their protocol. After exchanging the MACs and the randomness for each gate, the protocol can be executed. We omit the details of the protocol description here.

A.5. Acronyms

PRG	Pseudo-Random Generator

DES	Data Encryption Standard
VOPRF	Verifiable Oblivious Pseudo-Random Function
OPRF	Oblivious Pseudo-Random Function
PRF	Pseudo-Random Function
UC	Universal Composability
ZK	Zero-Knowledge
MPC	Multi-Party Computation
SFE	Secure Function Evaluation
OT	Oblivious Transfer
PAKE	Password Authenticated Key Exchange
aPAKE	asymmetric Password Authenticated Key Exchange
LWE	Learning With Errors
DDH	Decisional Diffie-Hellman Assumption
CDH	Computational Diffie-Hellman
NIST	National Institute of Standards and Technology
RSA	Rivest Shamir Adleman
PKI	Public-Key Infrastructure
CRS	Common Reference String
PPT	Probabilistic Polynomial Time
AES	Advanced Encryption Standard
ROM	Ranom Oracle Model
QROM	Quantum-accessible Random Oracle Model
AKE	Authenticated Key Exchange
DOS	Denial of Service
PRP	Pseudo-Random Permutation
MAC	Message Authentication Code
IETF	Internet Engineering Task Force

LAN	Local Area Network
WAN	Wide Area Network
SIS	Short Integer Solution
TLS	Transport Layer Security
HTTPS	Hypertext Transfer Protocol Secure
DLOG	Discrete Logarithm
q-**DHI**	Decisional q-Diffie-Hellman Inversion Problem