Composable Oblivious Pseudo-Random Functions via Garbled Circuits

Sebastian Faller[1,2][0009−0005−4126−3098], Astrid Ottenhues[3,4][0009−0007−3082−216X], and Johannes Ernst[5][0009−0001−3475−819X]

[1] IBM Research Europe, Switzerland
[2] ETH Zurich, Switzerland; sebastian.faller@ibm.com
[3] KASTEL Security Research Labs, Karlsruhe, Germany
[4] Karlsruhe Institute of Technology; astrid.ottenhues@kit.edu
[5] University of St. Gallen, Switzerland; johannes.ernst@unisg.ch

Abstract. Oblivious Pseudo-Random Functions (OPRFs) are a central tool for building modern protocols for authentication and distributed computation. For example, OPRFs enable simple login protocols that do not reveal the password to the provider, which helps to mitigate known shortcomings of password-based authentication such as password reuse or mix-up. Reliable treatment of passwords becomes more and more important as we login to a multitude of services with different passwords in our daily life.
To ensure the security and privacy of such services in the long term, modern protocols should always consider the possibility of attackers with quantum computers. Therefore, recent research has focused on constructing post-quantum-secure OPRFs. Unfortunately, existing constructions either lack efficiency, or they are based on complex and relatively new cryptographic assumptions, some of which have lately been disproved.
In this paper, we revisit the security and the efficiency of the well-known "OPRFs via Garbled Circuits" approach. Such an OPRF is presumably post-quantum-secure and built from well-understood primitives, namely symmetric cryptography and oblivious transfer. We investigate security in the strong Universal Composability model, which guarantees security even when multiple instances are executed in parallel and in conjunction with arbitrary other protocols, which is a realistic scenario in today's internet. At the same time, it is faster than other current post-quantum-secure OPRFs. Our implementation and benchmarks demonstrate that our proposed OPRF is currently among the best choices if the privacy of the data has to be guaranteed for a long time.

Keywords: Oblivious Pseudo-Random Function · Garbled Circuits · Post-Quantum Cryptography · Universal Composability

1 Introduction

An Oblivious Pseudo-Random Function (OPRF) is a two-party protocol for obliviously evaluating a Pseudo-Random Function (PRF), which is a function that outputs a pseudorandom value. One party (the server) holds the key k and

the other party (the user) has the input p. The goal is that the user does not learn anything about the server's key k while the server does neither learn anything about the user's input nor the output. This interaction is shown in Figure 1.

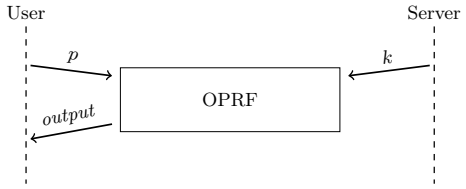

Fig. 1: Sketch of the Oblivious Pseudo-Random Function (OPRF) functionality.

OPRFs lie at the heart of many privacy-preserving protocols. To illustrate the importance of secure OPRF protocols we elaborate on some examples: Private set intersection [36] for instance allows two users to find out which contacts they both have in common, without revealing the full list of their contacts to each other or a service provider. The OPRF allows one party to hide its input while still computing some fingerprints of its elements for the other party. PrivacyPass [25] allows users to bypass subsequent Captchas (after solving the first one), while preventing tracking of the users. In this use-case, the OPRF is used for letting a user retrieve unlinkable tokens after solving a Captcha. OPAQUE [35] enables password-based authentication while hiding the password from the server, which alleviates many of the known problems of passwords. In OPAQUE, the OPRF is used to turn a low entropy password into a high entropy secret while completely hiding the password from the server. At the time of writing, OPAQUE is in the process of being standardized by the IETF[6]. One can see that all these protocols crucially rely on an OPRF as a cryptographic building block.

The above-mentioned protocols are designed to run in today's Internet, where many protocols run concurrently and are used as building blocks for other protocols. In these complex environments, attackers may be able to gain information by maliciously relaying messages from different sessions, or by otherwise making different protocol executions interfere. One of the most common approaches to construct *composable* protocols that keep their security guarantees in these complex situations was proposed in [16], called *universal composability*. Subsequently, this concept has also been applied to formalize and construct *composable* OPRFs [33–35]. Formally proving the security of an OPRF protocol in a model that guarantees composability is an important aspect of ensuring the security of a protocol in reality.

The security of most existing OPRF constructions is based on concrete hardness assumptions. A disadvantage of this is that any concrete assumption may be broken, which would then break the corresponding OPRF. On the other hand, if a protocol relies on more general assumptions such as secure symmetric encryption, one can easily switch from one symmetric encryption scheme to

[6] https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-09.html

another in case new attacks render the former one insecure. This makes generic, versatile OPRFs an important goal.

OPRFs were intensively studied in the literature and by now, classical, i.e., Discrete Logarithm (DLog) based protocols are amazingly efficient and enjoy a rich set of additional properties. Nonetheless, it still remains a challenge to construct similarly efficient and versatile OPRFs in the presence of adversaries with quantum computers. In their vision paper, Kampanakis et al. [37] identify the research on post-quantum (pq) OPRFs as a highly relevant research area for post-quantum migration, in particular because of the importance of OPRFs for pq anonymous authentication protocols. In this work, we focus on *presumably* pq-secure protocols, i.e., protocols that can be instantiated from pq-assumptions. This does not necessarily mean that the security proof considers quantum attackers.

When OPRFs are used in practice efficiency is a major concern. Therefore, it is important that OPRF proposals are accompanied by an implementation to analyze their efficiency and compare them to related results. A too slow OPRF in PrivacyPass [25] or OPAQUE [35] can significantly disturb user experience during web-browsing or authentication. Thus, improving the efficiency of pq OPRFs is a decisive objective, as the current (presumably) pq-secure OPRFs still do not match the classical constructions in terms of efficiency.

All together these motivations raise the following question:

Can we obtain an efficient, composable, and presumably post-quantum secure oblivious pseudo-random function constructed from generic techniques?

We answer this question in the affirmative. We show how to adapt an OPRF protocol from [43] based on Yao's garbled circuits such that this adapted version can be proven to be secure in the universal composability model of [16]. Because both garbled circuits and oblivious transfer can be instantiated from pq-secure primitives, our protocol is presumably pq-secure. We demonstrate its concrete efficiency via detailed benchmarks of our implementations.

1.1 Contribution

Our work on answering the above question is based on different areas, wherefore our contribution is threefold. We give a brief technical overview for each part. It can be summarized as follows:

1. We use the Multi-Party Computation (MPC) technique of Garbled Circuits to construct an OPRF protocol and prove its security in the Universal Composability (UC) framework against malicious users and semi-honest servers.
2. We implemented two versions of our protocol and compare it to other state-of-the-art protocols in extensive performance tests.
3. We compare two different approaches from the literature of defining OPRF security in the UC framework and show that one of them is strictly stronger and that it cannot be achieved by a large class of protocols.

(1) *Construction and Proof of OPRFs via Garbled Circuits.* By now multiple presumably post-quantum OPRF protocols have been proposed [1,2,11,12,30,31]. Some are based on new cryptographic assumptions and some have been broken. Therefore, an OPRF protocol that relies on generic and well studied building blocks is very desirable. The idea of using generic MPC techniques such as Garbled Circuits (GCs) to construct OPRFs has first been described in [43]. As we will argue, the protocol described in [43] does not satisfy the strong OPRF security definition of [34,35]. Our first contribution is that we show how a modification of the protocol from [43] can be proven secure in the model of [34,35] assuming semi-honest corruption of the server and malicious users. GCs have been optimized intensively [7,39,47], such that they have become efficient for computing functions that have a small representation as a boolean circuit. Furthermore, [13] shows that GCs can be proven secure against quantum attackers in a certain model [10], if instantiated with appropriate building blocks. As Oblivious Transfer (OT) and symmetric primitives can be instantiated in many different ways, the security of GCs does not depend just on a single hardness assumption—that might or might not be broken in the future. Because of these advantages of GCs, we followed an idea from [43] to construct OPRFs via GCs that can be sketched as follows: If a server and a user participate in a secure two-party computation, where the jointly evaluated circuit is a PRF, the resulting protocol is an OPRF. However, this construction does not yet achieve composability which is one of our main goals. To get security in the UC framework, we additionally introduce two hash functions, wich will be modeled as Random Oracles (ROs), following a general idea from [35]. The ROs are crucial for the security proof in the UC framework. We prove security assuming semi-honest servers and malicious users. We will elaborate further on this in Section 3.2. Because we prove security in the UC framework, the protocol can be securely used—even in parallel or concurrently—with itself or with other protocols.

(2) *Implementation and Benchmarks of our OPRF Protocol.* We implemented our protocol twice to compare its performance to the current state-of-the-art protocol, *2HashDH*, by [33–35], the lattice-based protocol by [2], and the isogeny-based protocol by [31]. The first implementation is in a C++ framework, called *EMP-Toolkit*[7], which offers most known optimizations for GCs. We also implemented the protocol with PQ-MPC[8]. This framework builds upon EMP-Toolkit and implements a garbling scheme that was proven secure by [13] in a model that considers powerful quantum adversaries [10]. We chose the Advanced Encryption Standard (AES) as the concrete instantiation of the PRF. We compared our implementations to an implementation of 2HashDH [35] building upon OPENSSL[9] and to a simplified implementation of the lattice-based protocol of [2]. We assess the efficiency of the implementations in terms of running time and network traffic. We performed our experiments on a conventional consumer laptop and measured

[7] https://github.com/emp-toolkit/emp-tool
[8] https://github.com/encryptogroup/PQ-MPC
[9] https://www.openssl.org/

the running time over the local network interface as well as over a simulated Wide Area Network (WAN). The experiments show that our protocol is much faster than the lattice-based construction of [2] and the isogeny-based protocols of [31]. Further, the experiments show that 2HashDH by [33–35] is still about 50 times faster than our construction and requires less than 100 B of communication. However, with a running time of about 22 ms and traffic of about 250 kB our protocol is still in a reasonable efficiency range. Considering the benchmark results, we see GC-based OPRFs as promising candidates for practically efficient OPRFs that are secure in the presence of adversaries with quantum computers.

(3) *Comparison of Different OPRF Functionalities.* In the literature on composable OPRFs, one can find two different approaches defining OPRFs in the UC framework. We compare a definition from [33–35] and an approach from [14]. We are the first to show that the first one is strictly stronger. This justifies our use of the stronger definition from [33–35] throughout this paper. We show that the plain protocol from [43] does not satisfy the stronger definition from [33–35] and we further show that it is impossible to prove a huge class of protocols secure under the stronger definition from [33–35] in the Non-Programmable Random Oracle Model (NPROM). The impossibility justifies our approach to achieving UC-security for the OPRF from [43] and it rules out the UC-security of a variety of constructions, including [12, Sec. 8] and [31].

1.2 Related Work

Started by [28], there is an ongoing research-line on OPRF protocols in which most protocols are based on the DLog or the integer factorization problem. This renders them vulnerable to potential quantum attacks. However, recent works focused on OPRFs based on presumably pq-secure assumptions. The first lattice-based construction was proposed by [2]. However, prohibitively large parameters must be chosen and expensive lattice-based zero-knowledge proofs are used. Additionally, the security analysis does not consider composition as our analysis does. A more efficient lattice-based OPRF was proposed in [1]. It uses FHE to evaluate the *Dark Matter weak PRF* proposed by [11]. Boneh at al. [11] also proposed an OPRF using the same weak PRF but instead of FHE they used MPC with preprocessing. This line of work was continued by Dinur et. al [26] who use secret-sharing-based MPC to evaluate a PRF that also builds on the modulus-switching idea from [11]. All those works have in common that they rely on the relatively new Dark Matter weak PRF. Although it is a very promising weak PRF candidate that currently receives a lot of attention from the research community, it is arguably not as thoroughly understood as well-established symmetric primitives like AES. Further, none of these works considers security in the UC framework which is crucial to applications like OPAQUE [35]. Two isogeny-based constructions were proposed by [12]. The authors estimate 424 kB of communication for the other protocol. However, it is not clear if the chosen parameters are sufficient or if bigger parameters are necessary to achieve a secure

protocol, see [4,20,21]. Also there is no UC proof for this OPRF. A second isogeny-based OPRF is proposed in the same work [12]. However, the construction was broken by [5], even before the underlying SIDH assumption was recently broken by [20]. Further, there is ongoing work on how to fix the shortcomings of the broken construction [4,31]. Another approach is to combine the PRF based on the Decisional Shifted Legendre Symbol Problem (DSLS) [24], which is presumably pq-secure, with a protocol allowing secure function evaluation over $\mathbb{F}_p$ for $p > 2$. The construction is proposed in [45] but no proof of security is given. A second drawback is that the pseudo-randomness of the Legendre symbol with hidden shift is not a standard assumption. There has been some work on the cryptanalysis of the assumption [9]. But one might be more confident in generic assumptions, e.g. OT or the existence of PRFs—like we use in our construction—because they are well-studied and there are several concrete instantiations.

To the best of our knowledge, the authors of [43] were the first proposing an OPRF construction from generic building blocks. They suggest realizing an OPRF by using GCs to evaluate the circuit of a PRF. The privacy requirement for the OPRF is satisfied as the GC protocol guarantees the privacy of inputs. A formal proof of security is not given in [43]. However, the work refers to the general proof for garbled circuit security in the presence of active adversaries of [40]. However, this proof analyzes very costly cut-and-choose techniques that make the garbling scheme rather impractical. The simulation-based proof uses the framework of [15] that even considers a weak form of composition. Note that the provided guarantees are not as strong as in the UC definition from [35]. In [38] a different approach is chosen. The authors use efficient OT extensions, introduced in [32], to instantiate something close to an OPRF protocol. The defined security notion is called batched related-key OPRF (BaRK-OPRF). This notion is related to usual OPRFs but it is not equivalent. BaRK-OPRF has the drawback that each PRF value is computed under a different key. While this limitation is not problematic for their use case of private set-intersection, it is not clear how to instantiate e.g. asymmetric Password Authenticated Key Exchange (aPAKE) [35], Password-Protected Secret Sharing (PPSS) [34], or distributed Single Sign On (SSO) [6] with BaRK-OPRF as these protocols require that the PRF is evaluated under the same key. The security is analyzed in a stand-alone simulation-based model, assuming server *and* client to be semi-honest, while our protocol only assumes semi-honest servers but allows malicious clients. A similarity between our protocols is that both rely only on the security of OT and symmetric cryptography. We summarized the above discussion in Table 1. For a more thorough discussion of OPRFs, we refer to [19].

2 Preliminaries

Pseudo-Random Functions. A Pseudo-Random Function (PRF) is a function that produces "random looking" output values. More precisely, the function is indexed by a key k and takes inputs x. If the key is chosen uniformly at random, the output $\mathsf{F}_k(x)$ is indistinguishable from a random value. The security is defined via a Probabilistic Polynomial Time (PPT) distinguisher $\mathcal{D}$ that either gets oracle

Protocol	UC Secure	(presum.) pq Secure	Pract. Efficient	Adv. Model	Assumption
Our work	✓	✓(OT)	✓	SH	OT, symm. crypto.
2HashDH [35]	✓	×	✓	M	om-DH
Plain Garbled Circuits [43]	×	✓(OT)	✓	SH	OT, symm. crypto.
BaRK-OPRF [38]	×	✓(OT)	✓	SH	OT, symm. crypto.
Lattice-based [2]	×	✓	×	M	RLWE, 1D-SIS
TFHE-based [1]	×	✓	×	SH	Dark-Matter wPRF
NR Isogeny-based [12]	×	✓	(✓)	SH	CSIDH
OPUS [31]	×	✓	×	SH	CSIDH,CSI-FiSh
Legendre-based [24, 30]	×	✓	?	?	DSLS
MPC w. preprocessing [11, 26]	×	✓	(✓)	SH	Dark-Matter wPRF

Table 1: Overview of related protocols. We write (✓) if we have not implemented the protocol in this work but an implementation will likely be efficient. We write ✓(OT) if a protocol is presumably pq-secure, as long as a pq-secure OT is used. We write M if the protocol is secure against malicious adversaries and SH if it is secure against semi-honest adversaries.

access to $\mathsf{F}_k(\cdot)$ for randomly chosen $k \in \{0,1\}^m$ or to a truly random function RF. The goal of $\mathcal{D}$ is to tell those situations apart.

Definition 1 (PRF & PRP). Let $\mathsf{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^l$ be a function family such that there is a polynomial-time algorithm that takes $k \in \{0,1\}^m$ and $p \in \{0,1\}^n$ and outputs $\mathsf{F}_k(p) \in \{0,1\}^l$. Let $p_0 := \Pr\left[\mathcal{D}^{\mathsf{F}_k(\cdot)}(1^\lambda) = 1\right]$ and $p_1 := \Pr\left[\mathcal{D}^{\mathsf{RF}(\cdot)}(1^\lambda) = 1\right]$, where the probabilities are taken over random choices of $k \in \{0,1\}^m$ and $\mathsf{RF} \in \{f : \{0,1\}^n \to \{0,1\}^l\}$. We say F is a *pseudo-random function* if the advantage $\mathrm{Adv}^{\mathsf{PRF}}_{\mathsf{F}}(\mathcal{D}, \lambda) := |p_0 - p_1|$ is negligible for every PPT distinguisher $\mathcal{D}$. If F_k is indistinguishable from a random permutation $\mathsf{RF} \xleftarrow{\$} \mathcal{S}_n$ then we say F is a *pseudo-random permutation.*

Oblivious Pseudo-Random Functions. A conventional PRF must be evaluated by a single party, which knows k as well as p. An OPRF for a certain PRF consists of two parties that interact to jointly compute an output of the PRF. One party—the server—holds the key k of the PRF and the other party—the user—holds the input value p. In the end, the user learns the output value $y = \mathsf{F}_k(p)$, 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 p.

The ROM and NPROM for UC. A random oracle $H\colon A \to B$ maps elements from a set A to elements of a set B. If H receives an input query $x \in A$ for the first time, it outputs a uniformly random drawn value $y \in B$ and stores the tuple $\langle x, y\rangle$. If H receives the query x again, it outputs y. To not clutter notation too much, we will notate the random oracle in our work like a "conventional hash function" instead of an ideal functionality.

In [41], the NPROM is defined as a variant of the UC framework of [16]. We provide a short intuition to the UC framework in Appendix B.1. In contrast to the original UC framework, each machine—including the environment machine—gets access to an oracle $\mathcal{O}$. The oracle is a random oracle in the sense that it answers queries of the form $x \in \{0,1\}^*$ with a uniformly random $y \in \{0,1\}^l$, where $l \in \mathbb{N}$ is fixed, and it records the tuple $\langle x, y \rangle$. If x is queried again, $\mathcal{O}$ answers again with y. The difference to a normal random oracle described before is that $\mathcal{O}$ is *not* a hybrid functionality. In particular, a simulator in the UC experiment has no way of influencing the output of $\mathcal{O}$. More precisely, we write $M^{\mathcal{O}}$ to denote that a machine M gets an oracle input tape, where M can write queries to x and an oracle output tape, where M receives the answers from $\mathcal{O}$. We say a protocol π UC-emulates a protocol ϕ in the NPROM if $\mathrm{EXEC}_{\pi^{\mathcal{O}},\mathcal{A}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}} \overset{c}{\approx} \mathrm{EXEC}_{\phi^{\mathcal{O}},\mathcal{S}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}}$.

Oblivious Transfer. In its simplest form, OT [44] allows a sender to transfer one of two messages to a receiver (1-out-of-2 OT). The receiver can choose which message it wants. The security guarantee for the receiver is that the sender does not learn anything about the choice of the receiver. The security guarantee for the sender is that the receiver does not learn anything about the message that was not chosen. Looking ahead, OT allows the evaluator of a Garbled Circuit (GC) to get the wire labels for their input, without leaking the input to the GC creator.

Garbled Circuits. Garbled circuits are a technique for secure two-party computation, which allows two parties to jointly evaluate any boolean circuit in a secure way. GCs ensure that the input of each party remains hidden from the other party. The original construction offers only security against a semi-honest garbler. The garbler could for example garble a different circuit, even one that leaks information about the evaluator's input. Several works improved the efficiency of GCs, most notably the techniques called *free-xor* [39] and *half-gates* [47].

Garbling Schemes. The authors of [8] defined an abstraction of the above-described GCs technique. They use a so-called *side-information function* Φ to parameterize their definitions. Given a circuit f, this function outputs certain information about the circuit. For this work, we will always assume that Φ outputs the entire circuit f, i.e., the PRF is public.

Definition 2 (Garbling Scheme [8, Sec. 3.1]). A *garbling scheme* is a tuple of a probabilistic garble algorithm Gb and deterministic algorithms En for encoding, De for decoding, Ev for garbled evaluation, and ev for "plain" evaluation, i.e., $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Let $f \in \{0,1\}^*$ be a description of the function that shall be garbled. The function $\mathsf{ev}(f, \cdot) : \{0,1\}^n \to \{0,1\}^m$ denotes the actual function, we want to garble, where $n \in \mathbb{N}$ and $m \in \mathbb{N}$ must be efficiently computable from f. On input f and a security parameter $\lambda \in \mathbb{N}$, the 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 an encoded output $Z = \mathsf{Ev}(F, X)$. String d describes a decoding function, $\mathsf{De}(d, \cdot)$, that maps an encoded output Z to a final output $z = \mathsf{De}(d, Z)$.

When we talk about the encoded input (sometimes we say *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 write $X[b]$ to denote the concatenation of the encodings of all bits in b. An execution of Yao's garbled circuits protocol is depicted in Figure 9 in Appendix B.

We require a garbling scheme to have *privacy* as defined in [8]. Intuitively, privacy means that anything that can be learned from the garbled circuit F, the input labels X, and the decoding information d, can also be learned from the output value y and the side-information $\Phi(f)$ alone. In particular, no efficient adversary can "break" the garbling scheme to get the input value of one of the parties. We give a formal definition of this notion in Appendix B.

3 Construction

In this section we present a protocol that UC-realizes $\mathcal{F}_{\text{OPRF}}$—the ideal OPRF functionality —under static malicious corruptions of users and static semi-honest corruptions of servers.

Adversarial Model. 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 a passive, i.e., semi-honest garbler, we will restrict our construction to these adversaries. This means that a corrupted garbler (in our case the server) follows the protocol honestly but tries to learn additional information from its view of the protocol execution. In Section 3.2 we discuss more reasons why evaluating a PRF with MPC is not sufficient to realize $\mathcal{F}_{\text{OPRF}}$ in the presence of a malicious server. However, we do allow malicious corruption of the evaluator (the user). Further, we assume static corruption. This means the adversary can only corrupt parties at the start of the protocol. 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.

Security Notion. We will not use the same formulation of the ideal OPRF functionality $\mathcal{F}^*_{\text{OPRF}}$ from [35] defined in Appendix B.1, but rather a slightly simplified version described in Figure 2. Note that $\mathcal{F}_{\text{OPRF}}$ does not capture the adaptive compromise of the server, as we only assume static corruption of servers. For the sake of simplicity, we also omit the prefixes used in $\mathcal{F}^*_{\text{OPRF}}$.

3.1 The main construction

Let $m, n \in \Omega(\lambda)$ and $F\colon \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ be a Pseudo-Random Permutation (PRP). In our implementation in Section 4, we instantiate the PRP with AES. We will garble the circuit C that describes F to construct our OPRF.

The user runs with $p \in \{0,1\}^*$ as input. This input is hashed to an n bit value, so we can use it as input to $\mathcal{C}$. Our construction involves two hash functions $H_1\colon \{0,1\}^* \to \{0,1\}^n$ and $H_2\colon \{0,1\}^* \times \{0,1\}^m \to \{0,1\}^l$, where $l \in \Omega(\lambda)$. We

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 $T_{sid}(i,p)$ is referenced below while it is undefined, draw $T_{sid}(i,p) \xleftarrow{\$} \{0,1\}^l$.
On (Init, sid) from S, if this is the first Init message for sid, set $\mathsf{tx}(sid) = 0$ and send $(\text{Init}, sid, \mathsf{S})$ 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.

Offline Evaluation: On $(\text{OfflineEval}, sid, i, p)$ from $\mathsf{P} \in \{\mathsf{S}, \mathcal{A}\}$, send $(\text{OfflineEval}, sid, T_{sid}(i,p))$ 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 $(\text{Eval}, sid, ssid, \mathsf{S}, p)$ from $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$, record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$ and send $(\text{Eval}, sid, ssid, \mathsf{P}, \mathsf{S})$ to $\mathcal{A}$.
- On $(\text{SndrCmplt}, sid, ssid)$ from S, increment $\mathsf{tx}(sid)$ or set to 1 if previously undefined, send $(\text{SndrCmplt}, sid, ssid, \mathsf{S})$ to $\mathcal{A}$. On ok from $\mathcal{A}$, send $(\text{SndrCmplted}, sid, ssid)$ to S.
- On $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ from $\mathcal{A}$, retrieve $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$, where $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$. Ignore this message if at least one of the following holds: (i) There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$, (ii) $i = \mathsf{S}$ but $\mathsf{tx}(sid) = 0$, (iii) S is honest but $i \neq \mathsf{S}$. Send $(\text{EvalOut}, sid, T_{sid}(i,p))$ to P. If $i = \mathsf{S}$ decrement $\mathsf{tx}(sid)$.

Fig. 2: The ideal functionality $\mathcal{F}_{\text{OPRF}}$ like in [35].

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 $\mathsf{F}_k(p) \coloneqq H_2(p, \mathcal{C}_k(H_1(p)))$.

In our description of the protocol depicted in Figure 3, the server garbles the circuit and the user evaluates the circuit. The user starts the execution of the protocol by hashing its input p. The obtained value $x = H_1(p)$ will be used as the user's input to the circuit. The user then requests a garbled circuit by sending $(\text{Garble}, sid, ssid)$ to the server. The server proceeds by generating the garbled circuit. In particular, it encodes its 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 to equip the user with the wire labels for its input $x = H_1(p)$. 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 $\rho = H_2(p, y)$. We describe the OPRF more precisely in Figure 4.

Theorem 1. *Let the garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ have* privacy*, as defined in Definition 4. Let $\mathcal{C}$ denote the boolean circuit of a PRP. Then* GC-OPRF *UC-realizes $\mathcal{F}_{\text{OPRF}}$ in the $\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{RO}}$-hybrid model.*

Proof sketch: The general strategy of the proof is as follows: First consider the case where both parties are honest. The simulator chooses a uniformly random

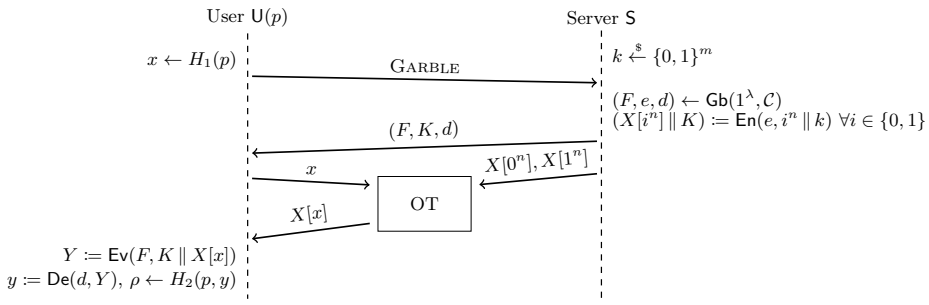

Fig. 3: Overview of GC-OPRF.

key k and runs the protocol like the real server would. The simulator does not get the user's input. But as it plays the role of $\mathcal{F}_{\text{OT}}$ it can report messages to the environment as if the user had given input to $\mathcal{F}_{\text{OT}}$. The simulator requests user output from $\mathcal{F}_{\text{OPRF}}$. We must argue why $\mathcal{F}_{\text{OPRF}}$ provides this output, i.e., why the counter is not exceeded. As both parties are honest, this is ensured by the simulator receiving a SndrCmplt message for the honest server. The OPRF output comes from $\mathcal{F}_{\text{OPRF}}$ and thus is a random value. The environment $\mathcal{E}$ can only distinguish it from the real output if it queries $H_2(p, \mathcal{C}_k(H_1(p)))$. We use the PRP property of $\mathcal{C}$ to argue that without any information about k, $\mathcal{E}$ sends this query with negligible probability. Next we consider the case where the user is maliciously corrupted. In contrast to the first case, $\mathcal{E}$ obtains the labels K of the key. Thus, $\mathcal{E}$ can query H_2 on p and $\mathsf{De}(d, \mathsf{Ev}(F, X[H_1(p)] \,\|\, K))$. Hence, the simulator must program H_2 accordingly on that input. To that end, Sim uses the RcvCmplt interface. However, that means the counter is decreased. We argue that $\mathcal{E}$ can only query $\mathsf{De}(d, \mathsf{Ev}(F, X[H_1(p)] \,\|\, K))$ if it received a garbling (F, K, d) and labels $X[H_1(p)]$ before. Sim produces this garbling if the counter was increased once. By the privacy of the garbling scheme, $\mathcal{E}$ gets only enough information to query H_2 on one such critical point for every garbling with labels that it obtains. Finally we consider the case where the server is passively corrupted. In this case, $\mathcal{E}$ learns the key $k_{\hat{\mathsf{S}}}$ of the server and can thus query $H_2(p, \mathcal{C}_{k_{\hat{\mathsf{S}}}}(H_1(p)))$. The simulator must detect these queries and use its OfflineEval interface to receive an output value ρ to program $H_2(p, \mathcal{C}_{k_{\hat{\mathsf{S}}}}(H_1(p))) \coloneqq \rho$. It is crucial that OfflineEval does not change $\mathcal{F}_{\text{OPRF}}$'s counter. Sim knows the key $k_{\hat{\mathsf{S}}}$, as we assume passive corruption of the server, i.e., the server does not maliciously choose some other key. Note that we used H_1 as a non-programmable but observable RO and H_2 as a programmable RO. We present the full proof in Appendix C.

3.2 Some Remarks on the Construction

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

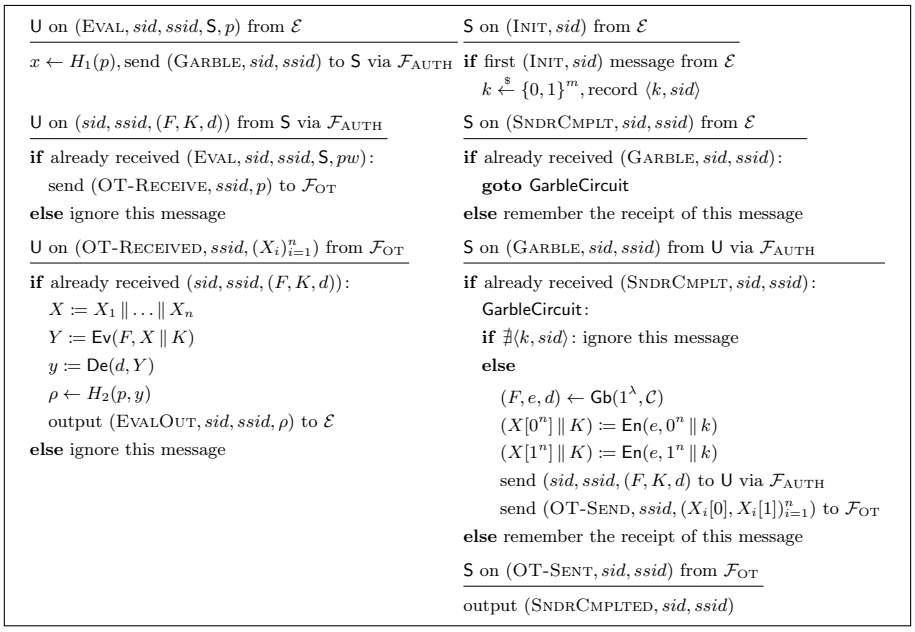

Fig. 4: The GC-OPRF construction in the $\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{AUTH}}$-hybrid model.

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 is 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 is secure against a semi-honest garbler. If the protocol would be implemented in a real-world scenario, it is a more realistic assumption that a server behaves in a semi-honest 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 are aware that malicious security is more desirable. However, techniques from the literature to achieve this are expensive in terms of running time and network traffic [40, 46]. If actively secure garbled circuits would be used, it might even be beneficial to switch roles. Then the user has to "invest" computation time on the creation of a garbled circuit, which decreases the threat of Denial of Service (DOS) attacks on the server. Note that actively secure garbling is still not enough to make our construction UC-realize $\mathcal{F}_{\text{OPRF}}$ in the presence of malicious servers, as we will detail below.

On the Need for Authenticated Channels. In the proof of security in Appendix C, we assume authenticated channels. This is necessary, as otherwise, we could not rely on the semi-honest nature of messages sent to the simulator. Assuming that the server behaves semi-honest, does not explicitly include the adversary. Thus, the adversary could still replace the honestly generated circuit from the

server with a malformed circuit. To avoid this problem, we assume authenticated channels, which prevent the adversary from replacing or injecting messages.

One could argue that the need for authenticated channels renders our construction impractical for many settings. For instance, if the OPRF is used for password-based authentication, one might not necessarily accept 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 Transport Layer Security (TLS) channel, and thus, at least the server is authenticated via digital certificates. We expect the security of our construction to hold even if only the server is authenticated. This does guarantee that the garbled circuit was generated by the party with which the user intends to communicate. Applications that build on top of TLS can thus make use of our OPRF protocol.

On security against semi-honest servers. Because our construction only provides security against semi-honest servers, let us discuss the implications of this in various use cases. When our OPRF protocol is used for secure password-based authentication as in OPAQUE, then a malicious server could learn the user's password. Note, however, that the main problem in practice is usually not that the service provider itself (e.g. email provider) is actively malicious, but rather that the server gets hacked and the data stolen. Multiple big tech companies[10] did log cleartext passwords. So if we trust the service provider not to be actively malicious, then our OPRF can be used to protect against such problems as inadvertent logging of cleartext passwords. Also, note that our protocol is secure against malicious users and, thus, one user cannot impersonate another user in an authentication setting like OPAQUE. In Privacy Pass[11] a malicious server may be able to learn the user-chosen token, which would allow them to track the user. This is probably an unacceptable risk for dissident Tor users, but for many others, it may be an acceptable risk. In Private Set Intersection (PSI) a malicious OPRF sender could learn all set elements of the other party, which clearly violates the security goals. The practical impact of this again depends on the trust relation between the two parties. If the OPRF sender is a somewhat trusted service provider then semi-honest security may be sufficient. Of course, it is better to choose a maliciously secure OPRF, if an efficient pq secure protocol is available, even though the semi-honest version provides sufficient security guarantees in many settings.

Challenges towards full malicious security. We like to highlight that for securing our construction against malicious servers it is likely necessary to employ actively secure garbled circuits (e.g. using cut-and-choose [40] or authenticated garbling [46]) but it is not sufficient, as the definition of $\mathcal{F}_{\text{OPRF}}$ has very strict security requirements. It guarantees that the output is uniformly random even if the

[10] https://www.zdnet.com/article/twitter-says-bug-exposed-passwords-in-plaintext, https://about.fb.com/news/2019/03/keeping-passwords-secure

[11] Privacy Pass requires a *verifiable* OPRF. One can follow the idea of [2] to achieve this using garbled circuits. We leave a proof of security of that to future work.

server is malicious, in particular, the server cannot force the output to be a specific value. For some protocols such as [35] this guarantee is necessary to prevent a Man-in-the-Middle attacker from impersonating an honest server. The first reason why evaluating a PRF with MPC is not enough to realize $\mathcal{F}_{\text{OPRF}}$ in the presence of a malicious server is that a PRF F can have *weak keys* (such as a key k, where $\mathsf{F}(k, x) = 0$, regardless of x). The function F can still be a secure PRF because the definition of a PRF only requires the output to look random if the key was chosen uniformly at random. However, the malicious server is not bound to choose the key randomly, but can deliberately choose a weak key, thereby violating the security guarantees of $\mathcal{F}_{\text{OPRF}}$. The second reason is more specific to the proof and the simulator. When the PRF key is the same in two sessions, then the same inputs must yield the same outputs. In the ideal world, the simulator determines a table via the RcvCmplt message, from which $\mathcal{F}_{\text{OPRF}}$ chooses the output value. To choose the correct table, the simulator has to be able to determine a value that uniquely identifies the key that the server used. In [34] this is achieved by letting the simulator include a DLOG trapdoor in the simulated messages of the honest user to the malicious server. In our case, as well as in other general MPC protocols, it is unclear how this unique identifier can be derived. The first reason shows that one would need to make stronger assumptions about the PRF to use it to build a maliciously secure OPRF. The second reason is an obstacle to the proof that may be solvable but would need new insights. Using covert security, which lies in between semi-honest and malicious security, also does not help, because a server choosing a different PRF key cannot be caught and, thus, has no incentive to be honest.

4 Comparison of Concrete Efficiency

As we were interested in the concrete efficiency of GC based OPRFs, we implemented the protocol from Section 3 in two versions and compared it to other OPRF protocols. We used AES-128 and AES-256 as instantiations for the circuit $\mathcal{C}$. For the first implementation, we leveraged a C++ framework, called EMP-Toolkit[7]. Because the available OT implementations from the EMP-Toolkit (and PQ-MPC) do not provide UC-security we opted for the most efficient OT protocol available in that library. We employed the Chou-Orlandi OT [22] protocol. Note that this is not a UC-secure OT protocol, as explained in the full-version of their paper [23]. It is also not post-quantum (PQ) secure as it relies on the GapDH assumption. If one wants to instantiate the OT using a protocol that is UC secure and plausible pq secure, one can use e.g. the OT proposed in [27]. The protocol from [27] is UC secure and can be instantiated under Learning Parity With Noise (LPN) in the CRS model. The protocol only requires two messages. Alternatively, one can employ the protocol proposed in [3]. It can also be instantiated under LPN and is secure in the Random Oracle Model (ROM). However, it is a four-message protocol, which might result in worse performance than [27]. We leave it to future work to implement a pq and UC secure OT protocol. We also used the pq-secure adaptation of the EMP-Toolkit, called PQ-MPC[8]. The security of this instantiation of garbled

circuits against quantum adversaries is proven in [13] in the Quantum-accessible Random Oracle Model (QROM) [10]. Our resulting OPRF implementation is presumably secure against quantum adversaries. As we are garbling a 128-bit-key AES circuit, we reach the same level of quantum security as defined in the NIST post-quantum competition as Security Strength Category 1[12]. Further, we implemented a version of the state-of-the-art OPRF protocol, 2HashDH, by [33–35]. Finally, we also compared the two former protocols to the lattice-based protocols of [2] and the isogeny-based protocol from [31]. We used the implementations that were provided by the respective works and run them on our machine. The main goal was to compare the concrete efficiency of different OPRFs on the same hardware. All source code described below can be found in the repository github.com/SebastianFaller/Composable-OPRF-via-Garbled-Circuits[13]. We also tried to run the provided implementation of [1]. But one OPRF evaluation did not finish within several hours on a normal laptop and therefore we did not benchmark the protocol extensively. This is not surprising as the benchmarks of [1] were performed on a server with 96 cores and over 700 GB RAM.

Benchmarks. We tested the implementations on a machine with an 11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz × 8 CPU. We made measurements on the local network interface, as well as a simulated WAN with limited bandwidth of 50 Mbit and a delay of 100 ms. We used the Linux tool `tc` for the latter. The WAN measurement simulates the situation where the server and user operate on machines in different countries or even continents. We measured the running time in milliseconds that each implementation needs from the invocation of a single 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 the user to the server and vice-versa. We summarized the results in Table 2.

Protocol	**Avg. Runtime (Local)** [ms]	**Avg. Runtime (WAN)** [ms]	**Network Traffic** [kB]	**UC**	**PQ**
Our work (AES-128, EMP-Tool)	19.92 ± 0.77	268.19 ± 19.42	232.71	×	×
Our work (AES-256, EMP-Tool)	26.53 ± 0.99	282.48 ± 26.04	299.78	×	×
Our work (AES-128, PQ-MPC)	47.12 ± 3.22	1696.91 ± 53.62	4746.13	×	✓
Our work (AES-256, PQ-MPC)	72.63 ± 4.51	2074.42 ± 22.98	6787.48	×	✓
2HashDH [35]	0.36 ± 0.13	201.88 ± 0.21	0.07	✓	×
Lattice VOPRF [2]	88512.92 ± 2079.35	95418.25 ± 989.30	513.25 ± 0.17	×	✓
OPUS [31]	11218.45 ± 61.98	35285.26 ± 36.50	24.70	×	✓

Table 2: Overview of the benchmark results. The protocol from [2] is a simplified version that does not include any Zero-Knowledge (ZK) proofs. The column **UC** marks if the implementation with their concrete building blocks is UC secure. Similarly, the column PQ indicates the same for plausible PQ security.

[12] https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[13] The benchmark results refer to the version of commit `ece1921`.

Running Time. We measured an average running time of 22.45 ms with a standard deviation of 1.73 ms for our GC-OPRF implementation with EMP-Toolkit and an average running time of 47.12 ms with a standard deviation of 3.22 ms for our GC-OPRF implementation with PQ-MPC. The performance of the PQ-MPC version of the protocol is still reasonable, as it is about twice as high as the running time for the classical circuit. The difference is a bit higher in the simulated WAN. This is most likely due to the bandwidth limitation, as the PQ-MPC version sends about 20 times as much data over the network as the EMP-Toolkit version.

Comparing with the other protocols in Table 2, we can see that our protocol is orders of magnitude more efficient than the competing pq-secure protocols. However, the elliptic curve protocol 2HashDH [35] is still more efficient than our protocol. Note that the comparison to [2] has to be taken with a grain of salt. On the one hand, their implementation omits all ZK proofs, which are necessary to make the protocol secure. These proofs would make the protocol even more impractical. On the other hand, the implementation was done with SageMath, while the implementations of our work, [35] and [31] were written in C++ and C, respectively. Further, the OPRF protocol from [2] is verifiable, which means that a server cannot arbitrarily choose new keys for each query. The other implementations do not have this property.

Network Traffic. Our EMP-Toolkit implementation sends 241.541 kB of data over the network, while the PQ-MPC version of our construction sends 4.746 MB. This huge difference comes from the fact that PQ-MPC uses a pq secure OT protocol based on FHE and does not use optimizations for garbled circuits, as e.g. free-xor [39] or half-gates [47]. While the Chou-Orlandi OT from the EMP-Toolkit implementation is based on GapDH, and thus, is certainly not pq secure, it is still plausible that all implemented garbled-circuit-optimizations from EMP-Toolkit are pq secure, as they do not involve DLog- or RSA-type assumptions. Also, note that the network traffic of all implementations except of [2] are constant values, while there are slight variations in the measurement for the protocol of [2]. This is because the transmitted value in the protocol is a random element in a cyclotomic ring. SageMath seems to automatically compress those elements if possible which leads to a varying size.

5 On the Ideal OPRF Functionality

In this section, we discuss two different UC-definitions of OPRFs and their relation. We show that the definition that we use is strictly stronger than the alternative formulation. It is important to note that security in the UC framework is always defined relative to an ideal functionality. Broadly speaking, a protocol UC-realizes a functionality if every attack that is possible against the real protocol is also possible against the ideal functionality. Thus, the security of the protocol highly depends on the definition of the ideal functionality.

5.1 Existing OPRF functionalities

There exist several descriptions of ideal OPRF functionalities in the literature [14,33–35]. What most of them have in common is that the ideal functionality

internally holds a truly random table of outputs from which the functionality delivers outputs to the user, when a protocol execution takes place. This ensures pseudo-randomness, as the output of the real-world protocol must be computationally indistinguishable from those random values. However, we are aware of one OPRF functionality that works differently. In [14] an ideal functionality is proposed that internally chooses a PRF key and delivers output values from one specific PRF to the user. We depict the differences in Figure 5. We will denote the first functionality that chooses the output values randomly by $\mathcal{F}_{\text{OPRF}}$ and the second functionality that is parameterized by a PRF $\mathsf{F}\colon \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^l$ as $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$. As a first step, we will show that the two definitions are indeed not equal in the sense that $\mathcal{F}_{\text{OPRF}}$ UC-realizes $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ but not vice-versa. We will also argue on a high level why a protocol using the weaker functionality $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ cannot e.g. control the number of password guesses after a server is compromised. This shows that password-based protocols such as aPAKE or PPSS must rely on the stronger functionality $\mathcal{F}_{\text{OPRF}}$. As a next step, we argue that a natural class of protocols cannot UC-realize $\mathcal{F}_{\text{OPRF}}$ in the NPROM from [41]. Finally, we argue that the garbled circuit-based OPRF proposed by [43] does not UC-realize $\mathcal{F}_{\text{OPRF}}$. This suggests the need to introduce a random oracle to our construction Section 3.1 and to program it in our proof of security in Appendix C.

5.2 Relation between $\mathcal{F}_{\text{OPRF}}$ and $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$

First, we establish that $\mathcal{F}_{\text{OPRF}}$ is stronger than $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$. Second, we show that $\mathcal{F}_{\text{OPRF}}$ can not be realized in the NPROM by a natural class of protocols. Lastly, we show that computing a PRF with garbled circuits alone cannot realize $\mathcal{F}_{\text{OPRF}}$. This gives a theoretical foundation for our proposed changes from Section 3, as we added a *programmable* random oracle to the construction from [43] to achieve the strong security notion of $\mathcal{F}_{\text{OPRF}}$. We defer the full proofs to Appendix D and give only intuition for each claim.

Claim 1. *$\mathcal{F}_{\text{OPRF}}$ UC-emulates $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$.*

Proof Sketch. The simulator just forwards all messages between $\mathcal{E}$ and the functionality. On corruption of the server, the simulator gets the key from $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$, which makes answering OfflineEval queries trivial.

Claim 2. *$\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ does not UC-emulate $\mathcal{F}_{\text{OPRF}}$.*

Proof Sketch. The second claim can be shown by contradiction. Assume there is a simulator. An environment can query arbitrarily many PRF values. If $\mathcal{E}$ corrupts the server after sufficiently many PRF queries, a simulator can find a key that matches all previously produced PRF outputs only with negligible probability. Thus, there cannot be such a simulator.

Note that we make use of $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$'s (Compromise, sid, S) interface that allows an adversary to get the key of the underlying PRF. Once the adversary has this key it can evaluate the PRF as often as it likes. Typically in password-based protocols, a user and a server execute the OPRF with the user's password as input. Now,

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

Initialization: For each value i and each session sid, an empty table $\boxed{F_{sid,i}(p),\ \text{for all inputs } p \in \{0,1\}^n}$ $\text{keys}(sid,i)$ is initially undefined. Whenever $\boxed{F_{sid,i}(p)}$ $\text{keys}(sid,i)$ is referenced below while it is undefined, draw $\boxed{F_{sid,i}(p) \xleftarrow{\$} \{0,1\}^l}$ $\text{keys}(sid,i) \xleftarrow{\$} \{0,1\}^m$.
On (Init, sid) from S, if this is the first Init message for sid, set $\mathsf{tx}(sid) = 0$ and send (Init, sid, 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.

Offline Evaluation: On (OfflineEval, sid, i, p) from $\mathsf{P} \in \{\mathsf{S}, \mathcal{A}\}$, send (OfflineEval, sid, $\boxed{F_{sid,i}(p)}$ $\text{keys}(sid,i)$) 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 (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 (Eval, $sid, ssid, \mathsf{P}, \mathsf{S}$) to $\mathcal{A}$.
- On (SndrCmplt, $sid, ssid$) from S, increment $\mathsf{tx}(sid)$ or set to 1 if previously undefined, send (SndrCmplt, $sid, ssid, \mathsf{S}$) to $\mathcal{A}$. On ok from $\mathcal{A}$, send (SndrCmplted, $sid, ssid$) to S.
- On (RcvCmplt, $sid, ssid, \mathsf{P}, i$) from $\mathcal{A}$, retrieve $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$, where $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$. Ignore this message if at least one of the following holds: (i) There is no record $\langle ssid, \mathsf{S}, \mathsf{P}, p\rangle$, (ii) $i = \mathsf{S}$ but $\mathsf{tx}(sid) = 0$, (iii) S is honest but $i \neq \mathsf{S}$. Send (EvalOut, sid, $\boxed{F_{sid,i}(p)}$ $\mathsf{F}_{\text{keys}(sid,i)}(p)$) to P. If $i = \mathsf{S}$ set $\mathsf{tx}(sid)$−−.

Fig. 5: Comparison between the ideal functionality $\mathcal{F}_{\text{OPRF}}$ inspired by [35] and the one inspired by [14]. Text in boxes is as in $\mathcal{F}_{\text{OPRF}}$, text in grey is inspired by [14]. Normal text is shared between both functionalities.

if an adversary gets unlimited PRF evaluations it can mount offline dictionary attacks against the password. This is also possible when using $\mathcal{F}_{\text{OPRF}}$ but the difference is that $\mathcal{F}_{\text{OPRF}}$ outputs one PRF-output per (OfflineEval, sid, p) message. Therefore, a protocol that uses $\mathcal{F}_{\text{OPRF}}$ as hybrid functionality can keep track of all password-guessing attempts of the adversary. The same is not possible with $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$, as the adversary can guess "locally" after it received the key.

Next, we formalize a natural class of protocols that cannot realize $\mathcal{F}_{\text{OPRF}}$.

Definition 3. We say a protocol has *reproducible output* if the following holds: In an execution with a passive, (i.e., semi-honest) adversary $\mathcal{A}$, every user U outputs—with overwhelming probability—the same output y when executing the protocol π on input p with a server S with fixed state k. In other words, the output of a protocol execution depends only on the user's input and on the server's internal state (and *not* e.g. on the user's internal state).

One can think of the server's state as the key of the server. But we cannot assume how this state may look for arbitrary protocol.

Claim 3. *Let π be a protocol that does not use any additional hybrid functionality and that has reproducible output. Then π does not UC-realize $\mathcal{F}_{\text{OPRF}}$ in the NPROM.*

Proof Sketch. The environment can execute a protocol run between a server and a user *internally* without the simulator being able to detect this. Then $\mathcal{E}$ can instruct two actual parties, i.e., parties that are not just internally run by $\mathcal{E}$, to let them execute the protocol. As the protocol π has reproducible output the output of the internal execution will be the same as the output of the external execution in the real-world experiment. By the definition of $\mathcal{F}_{\text{OPRF}}$ the output of the user in the external execution will be drawn uniformly random in the ideal-world experiment. Therefore, both outputs will not match with high probability.

We can use a very similar argument to prove a statement about the protocol proposed in [43]. We assume that the employed garbling scheme is correct and has privacy as given in Definition 4. Because it makes no difference to our argument, we can even assume that the employed garbling scheme from [43] uses a programmable random oracle.

Claim 4. *Executing garbled circuits to jointly compute a PRF F between a user and a server—as proposed in [43]—is not sufficient to UC-realize $\mathcal{F}_{\text{OPRF}}$ in the NPROM.*

Claim 3 and Claim 4 justify our method to achieve a UC-secure protocol from the protocol of [43]. Loosely speaking, they say that without exploiting the programmability of a random oracle, one cannot realize the strong OPRF notion of [35] that is used e.g. in OPAQUE or the PPSS scheme [34]. To overcome this, we added two random oracles and carefully programmed them in the UC-proof.

6 Conclusion

In this work, we investigated the security of a garbled-circuit-based OPRF in the UC-framework [16]. To realize an ideal OPRF functionality in the style of [33–35], we augmented the construction of [43] with two hash functions, of which the second was modeled as a programmable random oracle. The resulting protocol is secure assuming static passive corruptions of servers and malicious users. We implemented two prototypes of our protocol—one using the optimized garbling scheme from EMP-Toolkit and one using the post-quantum garbling scheme from PQ-MPC. Although both implementations use building blocks that are not proven to be UC secure, to the best of our knowledge, our implementation is the only presumably pq secure implementation that can be made UC secure by plugging in UC secure building blocks. We also implemented the state-of-the-art OPRF protocol 2HashDH by [33–35]. We compared the implementations to a simplified implementation of the lattice-based OPRF by [2] and the isogeny-based protocol from [31]. The experiments showed that our construction is significantly faster than the lattice-based and isogeny-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 22 ms

and around 250 kB network traffic. This indicates, that garbled-circuit-based OPRF protocols are very promising candidates for pq secure OPRFs. Finally, we investigated the theoretical differences between definitions of OPRFs in the UC framework. We compared the ideal functionalities in the style of [33–35] and a functionality in the style of [14] and showed that the functionality from [33–35] is strictly stronger. We show that a natural class of protocols cannot realize this strong functionality in the NPROM. We further show that the OPRF protocol from [43], cannot realize the strong functionality. The last two claims justify our approach of augmenting the protocol from [43] by a random oracle and programming it in the security proof.

Acknowledgements. We thank the LATINCRYPT 2023 anonymous reviewers for their valuable feedback, and espacially Octavio Pérez Kempner for the constructive and helpful support during the shepherding process. Astrid Ottenhues: This work was supported by funding by the German Federal Ministry of Education and Research (BMBF) under the projects "PQC4MED" (ID 16KIS1044) and "Sec4IoMT" (ID 16KIS1692), and by KASTEL Security Research Labs.

References

1. Albrecht, M.R., Davidson, A., Deo, A., Gardham, D.: Crypto dark matter on the torus: Oblivious PRFs from shallow PRFs and FHE. Cryptology ePrint Archive, Report 2023/232
2. Albrecht, M.R., Davidson, A., Deo, A., Smart, N.P.: Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In: PKC 2021, Part II. LNCS, vol. 12711. Springer, Heidelberg, Germany, Virtual Event
3. Barreto, P.S.L.M., David, B., Dowsley, R., Morozov, K., Nascimento, A.C.A.: A framework for efficient adaptively secure composable oblivious transfer in the ROM. Cryptology ePrint Archive, Report 2017/993
4. Basso, A.: A post-quantum round-optimal oblivious PRF from isogenies. Cryptology ePrint Archive, Report 2023/225
5. Basso, A., Kutas, P., Merz, S.P., Petit, C., Sanso, A.: Cryptanalysis of an oblivious PRF from supersingular isogenies. In: ASIACRYPT 2021, Part I. LNCS, vol. 13090. Springer, Heidelberg, Germany, Singapore
6. Baum, C., Frederiksen, T.K., Hesse, J., Lehmann, A., Yanai, A.: Proactively secure distributed single sign-on, or how to trust a hacked server. Cryptology ePrint Archive, Report 2019/1470
7. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, Berkeley, CA, USA
8. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS 2012. ACM Press, Raleigh, NC, USA
9. Beullens, W., Beyne, T., Udovenko, A., Vitto, G.: Cryptanalysis of the Legendre PRF and generalizations. IACR Trans. Symm. Cryptol. **2020**(1)
10. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: ASIACRYPT 2011. LNCS, vol. 7073. Springer, Heidelberg, Germany, Seoul, South Korea

11. Boneh, D., Ishai, Y., Passelègue, A., Sahai, A., Wu, D.J.: Exploring crypto dark matter: New simple PRF candidates and their applications. In: TCC 2018, Part II. LNCS, vol. 11240. Springer, Heidelberg, Germany, Panaji, India
12. Boneh, D., Kogan, D., Woo, K.: Oblivious pseudorandom functions from isogenies. In: ASIACRYPT 2020, Part II. LNCS, vol. 12492. Springer, Heidelberg, Germany, Daejeon, South Korea
13. Büscher, N., Demmler, D., Karvelas, N.P., Katzenbeisser, S., Krämer, J., Rathee, D., Schneider, T., Struck, P.: Secure two-party computation in a quantum world. In: ACNS 20, Part I. LNCS, vol. 12146. Springer, Heidelberg, Germany, Rome, Italy
14. Camenisch, J., Lehmann, A.: Privacy-preserving user-auditable pseudonym systems. In: 2017 IEEE European Symposium on Security and Privacy (EuroSP)
15. Canetti, R.: Security and composition of multi-party cryptographic protocols. Cryptology ePrint Archive, Report 1998/018
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. IEEE Computer Society Press, Las Vegas, NV, USA
17. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067
18. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC. ACM Press, Montréal, Québec, Canada
19. Casacuberta, S., Hesse, J., Lehmann, A.: SoK: Oblivious pseudorandom functions. Cryptology ePrint Archive, Report 2022/302
20. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: EUROCRYPT 2023, Part V. LNCS, vol. 14008. Springer, Heidelberg, Germany, Lyon, France
21. Chávez-Saab, J., Chi-Domínguez, J.J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: Sublinear Vélu quantum-resistant isogeny action with low exponents. Cryptology ePrint Archive, Report 2020/1520
22. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: LATINCRYPT 2015. LNCS, vol. 9230. Springer, Heidelberg, Germany, Guadalajara, Mexico
23. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. Cryptology ePrint Archive, Report 2015/267
24. Damgård, I.: On the randomness of Legendre and Jacobi sequences. In: CRYPTO'88. LNCS, vol. 403. Springer, Heidelberg, Germany, Santa Barbara, CA, USA
25. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. PoPETs **2018**(3)
26. Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In: CRYPTO 2021, Part IV. LNCS, vol. 12828. Springer, Heidelberg, Germany, Virtual Event
27. Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: EUROCRYPT 2020, Part II. LNCS, vol. 12106. Springer, Heidelberg, Germany, Zagreb, Croatia
28. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: TCC 2005. LNCS, vol. 3378. Springer, Heidelberg, Germany, Cambridge, MA, USA
29. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. In: Automata, Languages and Programming. Lecture Notes in Computer Science, Springer

30. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: ACM CCS 2016. ACM Press, Vienna, Austria
31. Heimberger, L., Meisingseth, F., Rechberger, C.: Oprfs from isogenies: Designs and analysis. Cryptology ePrint Archive, Paper 2023/639
32. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO 2003. LNCS, vol. 2729. Springer, Heidelberg, Germany, Santa Barbara, CA, USA
33. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: ASIACRYPT 2014, Part II. LNCS, vol. 8874. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C.
34. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). Cryptology ePrint Archive, Report 2016/144
35. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: EUROCRYPT 2018, Part III. LNCS, vol. 10822. Springer, Heidelberg, Germany, Tel Aviv, Israel
36. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC 2009. LNCS, vol. 5444. Springer, Heidelberg, Germany
37. Kampanakis, P., Lepoint, T.: Vision paper: Do we need to change some things? In: Security Standardisation Research. Springer Nature Switzerland, Cham
38. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: ACM CCS 2016. ACM Press, Vienna, Austria
39. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: ICALP 2008, Part II. LNCS, vol. 5126. Springer, Heidelberg, Germany, Reykjavik, Iceland
40. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: EUROCRYPT 2007. LNCS, vol. 4515. Springer, Heidelberg, Germany, Barcelona, Spain
41. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: CRYPTO 2002. LNCS, vol. 2442. Springer, Heidelberg, Germany, Santa Barbara, CA, USA
42. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: CRYPTO 2008. LNCS, vol. 5157. Springer, Heidelberg, Germany, Santa Barbara, CA, USA
43. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: ASIACRYPT 2009. LNCS, vol. 5912. Springer, Heidelberg, Germany, Tokyo, Japan
44. Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187
45. Seres, I.A., Horváth, M., Burcsi, P.: The legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications. Cryptology ePrint Archive, Report 2021/182
46. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: ACM CCS 2017. ACM Press, Dallas, TX, USA
47. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: EUROCRYPT 2015, Part II. LNCS, vol. 9057. Springer, Heidelberg, Germany, Sofia, Bulgaria

A Acronyms

AES	Advanced Encryption Standard
aPAKE	asymmetric Password Authenticated Key Exchange
CRS	Common Reference String
CSIDH	Commutative Supersingular Isogeny Diffie–Hellman
CSI-FiSh	Commutative Supersingular Isogeny-based Fiat-Shamir Signatures
DLog	Discrete Logarithm
DSLS	Decisional Shifted Legendre Symbol Problem
DOS	Denial of Service
FHE	Fully Homomorphic Encryption
GC	Garbled Circuit
LPN	Learning Parity With Noise
MPC	Multi-Party Computation
NIST	National Institute of Standards and Technology
NPROM	Non-Programmable Random Oracle Model
om-DH	One-More Diffie-Hellman
OPRF	Oblivious Pseudo-Random Function
OT	Oblivious Transfer
PPSS	Password-Protected Secret Sharing
PPT	Probabilistic Polynomial Time
PRF	Pseudo-Random Function
PRP	Pseudo-Random Permutation
PSI	Private Set Intersection
PQ	post-quantum
pq	post-quantum
RLWE	Ring Learning With Errors
RO	Random Oracle
ROM	Random Oracle Model
RSA	Rivest Shamir Adleman
SIDH	Supersingular Isogeny Diffie–Hellman
SSO	Single Sign On
TLS	Transport Layer Security
UC	Universal Composability
VOPRF	Verifiable Oblivious Pseudo-Random Function
WAN	Wide Area Network
ZK	Zero-Knowledge
1D-SIS	1-Dimensional Short Integer Solution Assumption

B Extended Preliminaries

Notation: We write λ for the security parameter. We always assume that all algorithms take λ as an implicit parameter. We call a probabilistic Turing machine Probabilistic Polynomial Time (PPT) if its running time is bounded by

a polynomial in λ. By $x \xleftarrow{\$} S$ we denote that x is chosen uniformly at random from the set S. We write $y \leftarrow A(x)$ to assign the output of the randomized algorithm A on input x to the variable y. We write $x \,\|\, y$ for the concatenation of the strings $x \in \{0,1\}^*$ and $y \in \{0,1\}^*$. We use $\mathcal{O}(\cdot)$, $o(\cdot)$, $\Theta(\cdot)$, $\Omega(\cdot)$, 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. More precisely, we call a function $f : \mathbb{N} \to \mathbb{R}$ negligible if for every $c \in \mathbb{N}$ there is an $N \in \mathbb{N}$, such that for all $n > N$ it holds that $|f(n)| < n^{-c}$. We call f *noticeable* if there exists $c \in \mathbb{N}, N \in \mathbb{N}$, such that for all $n > N$ it holds that $|f(n)| \geq n^{-c}$. Note that noticeable is a stronger notion than being non-negligible. We will call a protocol *presumably* pq-secure if it is secure against classical adversaries and completely relies on post-quantum assumptions such as certain lattice problems. On the other hand, we call a protocol pq-secure if its security proof considers quantum adversaries.

Security of Garbling Schemes [8] gave a game-based security definition, as well as a simulation-based security definition for the first two properties.We use only the simulation-based notions in this work.

Definition 4 (Privacy). [8, Sec. 3.4] For a simulator $\mathcal{S}$, we define the advantage of adversary $\mathcal{A}$ in the security experiment defined in Figure 6, as

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

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 $\mathrm{negl}(\cdot)$.

Game $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(\lambda)$

- The challenger C chooses a bit $b \in \{0,1\}$ uniformly at random.
- $\mathcal{A}$ sends a function $f : \{0,1\}^n \to \{0,1\}^*$ 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 $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f)$ and $X \coloneqq \mathsf{En}(e, x)$.
  Else the challenger calculates $y \coloneqq \mathsf{ev}(f, x)$ 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 $b' \stackrel{?}{=} b$.

Fig. 6: The simulation-based privacy game from [8, Fig. 5].

B.1 Universal Composability

Often, cryptographic protocols are not used in isolation but are combined to serve in a bigger context. However, the security of protocols is not always conserved under composition. For example, the parallel composition of two ZK protocols is in general not a ZK protocol [29]. Universal Composability (UC), introduced in [16] is a notion of security that tackles this problem. Protocols that are secure in the UC model can be composed while keeping their security as stated formally in the *composition theorem* [17, 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 proof. In the ideal world, we do not regard the actual protocol π 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 a subroutine in another protocol. In contrast to a 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. We will denote the view of the environment on this interaction by $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$. In the ideal world, the protocol parties are replaced by "Dummy-Parties", which relay the input from $\mathcal{E}$ to $\mathcal{F}$ and vice-versa. 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 $\mathcal{E}$ in a way that looks like the real-world execution. If no PPT environment can tell both worlds apart, the protocol π UC-realizes the ideal functionality $\mathcal{F}$, see [17, Def. 9]. We say a protocol π *UC-emulates* a protocol ϕ ($\pi \geq \phi$) if for all PPT adversaries $\mathcal{A}$ there is a PPT simulator $\mathcal{S}$, such that for all environment machines $\mathcal{E}$ it holds that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}} \overset{c}{\approx} \text{EXEC}_{\phi,\mathcal{S},\mathcal{E}}$. Let $\text{IDEAL}_{\mathcal{F}}$ denote the protocol that consists of a machine $\mathcal{F}$, the ideal functionality, and a Dummy-Party for each protocol party of π. We say a protocol π *UC-realizes* a functionality $\mathcal{F}$, if π UC-emulates $\text{IDEAL}_{\mathcal{F}}$, see [17, Definition 20].

Authenticated Channels In [17] authenticated communication is defined via an ideal functionality that closely resembles the one depicted in Figure 7. Note that Figure 7 is a bit simpler as the functionality from [17], as we do not deal with adaptive corruptions.

Oblivious Transfer We slightly adapt an ideal functionality introduced by [18] and used e.g. by [42]. It considers *one* sender and *one* receiver.

For the sake of clarity, we augment the original functionality from [18] with explicit messages allowing the sender to delay executions. 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 8. Further directly notate that the sender sends n pairs of messages and the receiver chooses n of these messages by sending a n-bit string. One can easily realize

this functionality with every protocol that already UC-realizes a 1-out-of-OT by simply executing the protocol n times in parallel. The protocol yields the output after all n sub-OTs are completed. The security follows directly from the UC-composition theorem [17, Theo.22].

Security of OPRFs We recall the security notion defined in [35]. The security is defined in the UC-framework. We describe the ideal functionality $\mathcal{F}^*_{\text{OPRF}}$ in Figure 10. We will write $\mathcal{F}^*_{\text{OPRF}}$ to distinguish this functionality, from the slightly simplified functionality $\mathcal{F}_{\text{OPRF}}$, which we introduce in Section 3.

The intuition of the functionality is that in each session multiple users interact with a server. A session is indexed by an id *sid*. 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 $(\text{EVAL}, sid, ssid, \mathsf{S}', x)$ to $\mathcal{F}^*_{\text{OPRF}}$. 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 does not know the value k but rather specifies the server that holds k. As we assume that an honest 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 an initially empty table and gets lazily filled with randomly drawn values.

A server can consent to the interaction with the user by sending $(\text{SNDRCMPLT}, sid, ssid)$ to $\mathcal{F}^*_{\text{OPRF}}$. Finally, the adversary can send $(\text{RCVCMPLT}, sid, ssid, \mathsf{U}, i)$ to $\mathcal{F}^*_{\text{OPRF}}$ to indicate that the user U can receive the requested output. However, $\mathcal{F}^*_{\text{OPRF}}$ 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 be chosen. If the adversary sends $(\text{RCVCMPLT}, sid, ssid, \mathsf{U}, \mathsf{S}')$, where S' is the server from the user's $(\text{EVAL}, sid, ssid, \mathsf{S}', x)$ 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 $\mathcal{F}^*_{\text{OPRF}}$ undermines the security of OPRF protocols realizing $\mathcal{F}^*_{\text{OPRF}}$. 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 $\mathsf{tx}(\cdot)$. With this counter, $\mathcal{F}_{\text{OPRF}}$ keeps track of the number of OPRF outputs that a server generates and the number of OPRF

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

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

Fig. 7: The ideal functionality $\mathcal{F}_{\text{AUTH}}$ from [17].

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

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

- Upon receiving a message $(\text{OT-SEND}, sid, (x_0^i, x_1^i)_{i=1}^n)$ from S, where each $x_0^i, x_1^i \in \{0,1\}^m$, record $\langle sid, (x_0^i, x_1^i)_{i=1}^n \rangle$. 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 $b \in \{0,1\}^n$ 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 $\langle sid, (x_0^i, x_1^i)_{i=1}^n \rangle$ 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 $\langle sid, (x_0^i, x_1^i)_{i=1}^n \rangle$ or $\langle sid, b \rangle$ is not recorded; Otherwise return $(\text{OT-RECEIVED}, sid, (x_b^i)_{i=1}^n)$ to R; Ignore further $(\text{OT-RECEIVED}, sid, \dots)$ messages from the adversary.

Fig. 8: Our ideal functionality $\mathcal{F}_{\text{OT}}$.

outputs from that server that is delivered to the user by the adversary. More precisely, every time the server consents to participate in an OPRF execution by sending $(\text{SNDRCMPLT}, sid, ssid, \mathsf{S})$, the counter $\mathsf{tx}(\mathsf{S})$ is incremented. If an output from S is delivered to a user by a $(\text{RCVCMPLT}, sid, ssid, \mathsf{U}, \mathsf{S})$ message to $\mathcal{F}^*_{\text{OPRF}}$, the counter $\mathsf{tx}(\mathsf{S})$ is decremented. If the counter is zero but output is request by a $(\text{RCVCMPLT}, sid, ssid, \mathsf{U}, \mathsf{S})$ message, $\mathcal{F}^*_{\text{OPRF}}$ ignores this message.

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

1. If the server i is corrupted. This represents 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 key.
3. A real-world adversary can just make up 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. We call them "virtual corrupt identities". For 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, in the sense that an adversary can now evaluate the PRF offline.

We also note that the ideal OPRF functionality from [34] assumes authenticated channels, while the ideal functionality from [35] dispenses with them.

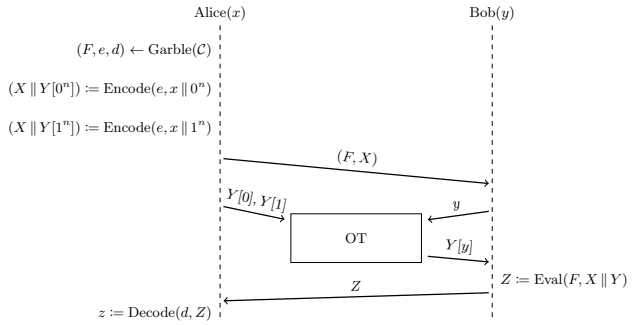

Fig. 9: Yao's garbled circuits protocol.

C Proving Security

In order to prove that the protocol GC-OPRF Figure 4 actually UC-emulates $\mathcal{F}_{\text{OPRF}}$ Figure 2 in the $\{\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model, we have to compare the views of two protocol executions. More precisely, for every PPT adversary $\mathcal{A}$ we must specify a PPT simulator Sim such that for every PPT environment $\mathcal{E}$ we have:

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

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

As discussed in [17], we will only consider a Dummy-Adversary $\mathcal{A}$. We construct the simulator as in Figures 11 to 15. For the sake of readability, we split the description of Sim into five figures. We write $\langle\cdot\rangle$ for records, made by the simulator. We denote parties with a hat, e.g. $\hat{\mathsf{P}}$, if it is clear from the context that they are corrupted. We 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.

Some Intuition on the Simulator Before we give a formal proof, we like to give some intuition on the simulator in Figures 11 to 15. First, note that in the formulation of the UC security experiment, 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 [16]. Nonetheless, we write in Figures 11 to 15 as if there were 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 $\{\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model, Sim has to simulate these functionalities.

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 (Init, *sid*) from S, if this is the first Init message for *sid*, set $\mathsf{tx} = 0$ and send (Init, *sid*, 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 (Compromise, *sid*, S) from $\mathcal{A}$, mark S as Compromised. If S is corrupted, it is marked as Compromised from the beginning. *Note: Message* (Compromise, *sid*, S) *requires permission from the environment.*

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

Evaluation:

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

Fig. 10: The ideal functionality $\mathcal{F}^*_{\text{OPRF}}$ from [35].

We note again that a random oracle is strictly speaking an ideal functionality, too. Thus, Sim must also answer queries to the random oracles H_1, and H_2. We omit the explicit notation of the simulation of $\mathcal{F}_{\text{AUTH}}$ queries for the sake of readability. 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 $\mathcal{F}_{\text{OT}}$. Then Sim receives a garbled circuit and input labels but for every input bit $x_i \in \{0,1\}$, Sim receives both labels $X_i[0]$ and $X_i[1]$, again because Sim simulates $\mathcal{F}_{\text{OT}}$. Sim requests an output for the user from $\mathcal{F}_{\text{OPRF}}$. Now, $\mathcal{F}_{\text{OPRF}}$ 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 $\mathcal{F}_{\text{OPRF}}$. Hence, the simulator must ensure that the output generated by $\mathcal{F}_{\text{OPRF}}$ and $H_2(p, y)$ coincide for values of p and y that can occur in the execution of the protocol. Sim can query output from $\mathcal{F}_{\text{OPRF}}$ but this has to be done carefully as $\mathcal{F}_{\text{OPRF}}$ 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 $y = \mathcal{C}_k(H_1(p))$. 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. The environment would notice a difference to the real-world execution as soon as it instructs the adversary to make the user output two output values from two different virtual corrupt identities by sending $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ and $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, i')$ in two executions with the same server input but $i \neq i'$.

One might be tempted to try the other extreme instead. What happens if the simulator uses a completely new identity i for every new query? By that we mean the following: Sim can query PRF output for a server identity i from $\mathcal{F}_{\text{OPRF}}$ if this i is no identity of an actual server of the session, as defined by *OfflineEval* point (iii) in Figure 2. These virtual corrupt identities do not have a ticket counter. By using sending $(\text{OfflineEval}, sid, i, p)$ to $\mathcal{F}_{\text{OPRF}}$, the simulator receives the entry $T_{sid}(i, p)$ from $\mathcal{F}_{\text{OPRF}}$'s table. This identity i must not correspond to an existent server in that session sid. Why can Sim not create a new such virtual corrupt identity for every H_2 query it receives? One can easily see that Sim would need to use the same virtual corrupt identity again when simulating a protocol

execution. = Sim has to send $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, i)$, where i was used to program the output of H_2 for the corresponding input. But by the definition of the ideal functionality, Sim does not learn the input of the user. Consequently, programming of H_2 must somehow depend on the identity of the server.

Our strategy for programming $H_2(p, y)$ is the following: If Sim receives a query, it looks up the corresponding H_1 query $H_1(p) = x$. 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 x for the circuit. Now, it checks if there either was an honest server or a corrupted server, such that $y = \mathcal{C}_k(x)$ holds for the key k of one of the servers. For an honest server, Sim requests the output value from $\mathcal{F}_{\text{OPRF}}$ by sending a RcvCmplt message and for a corrupted server, Sim requests the output value from $\mathcal{F}_{\text{OPRF}}$ with an 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 $(\text{Eval}, sid, ssid, \mathsf{S}, p)$ messages to U. User U transmits this message to $\mathcal{F}_{\text{OPRF}}$ and outputs $(\text{EvalOut}, sid, ssid, \rho)$ to $\mathcal{E}$.
- Honest server S:
  - $\mathcal{E}$ sends (Init, sid) to S. Server S transmits this message to $\mathcal{F}_{\text{OPRF}}$ who sends $(\text{Init}, sid, \mathsf{S})$ to $\mathcal{A}$.
  - $\mathcal{E}$ sends $(\text{SndrCmplt}, sid, ssid)$ to S. Server S forwards this message to $\mathcal{F}_{\text{OPRF}}$. The functionality $\mathcal{F}_{\text{OPRF}}$ forwards this message to $\mathcal{A}$. If $\mathcal{A}$ approves with ok, $\mathcal{F}_{\text{OPRF}}$ sends $(\text{SndrCmplted}, sid, ssid)$ to the server.
- Dummy adversary $\mathcal{A}$:
  - The environment can send $(\text{Garble}, sid, ssid)$, and $(\text{OT-Receive}, ssid, x)$ 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 $(sid, ssid, (F, K, d))$, and $(\text{OT-Send}, ssid(X_i[0], X_i[1])_{i=1}^n)$ 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 $(\text{OT-Sent}, ssid)$, $(\text{OT-Received}, ssid)$ to $\mathcal{A}$. The adversary $\mathcal{A}$ will send these messages to Sim, acting as the 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:

- $(\text{EvalOut}, sid, ssid, \rho)$ from U as response to an $(\text{Eval}, sid, ssid, \mathsf{S}, p)$ message.
- $(\text{SndrCmplted}, sid, ssid)$ from S when $\mathcal{A}$ approved a $(\text{SndrCmplt}, sid, ssid)$ message by sending ok.

- (GARBLE, sid, $ssid$) from $\mathcal{A}$ when $\mathcal{A}$ acts as the server and receives this message, formatted as being sent from a user via $\mathcal{F}_{\text{AUTH}}$.
- (sid, $ssid$, (F, K, d)) from $\mathcal{A}$ when $\mathcal{A}$ acts as a user and receives this message, formatted as being sent from a server via $\mathcal{F}_{\text{AUTH}}$.
- (OT-SEND, $ssid$) from $\mathcal{A}$ when a server sends OT-input to Sim, who acts as $\mathcal{F}_{\text{OT}}$.
- (OT-RECEIVE, $ssid$) from $\mathcal{A}$ when a user sends choice bits to Sim, who acts as $\mathcal{F}_{\text{OT}}$.
- (OT-SENT, sid) from $\mathcal{A}$ when $\mathcal{A}$ acts as server and sent (OT-SEND, sid, $(X_i[0], X_i[1])_{i=1}^n$) to Sim before. Sim acts as $\mathcal{F}_{\text{OT}}$.
- (OT-RECEIVED, sid, $(X_i)_{i=1}^n$) from $\mathcal{A}$ when $\mathcal{A}$ acts as user and sent (OT-RECEIVE, $ssid$, x) to Sim before. Sim acts as $\mathcal{F}_{\text{OT}}$.
- 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 (GARBLE, sid, $ssid$) that is formatted exactly like a (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 (GARBLE, sid, $ssid$) when an honest user is invoked by $\mathcal{E}$ with (EVAL, sid, $ssid$, S, p), 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 the user is corrupted, (3) only the server is corrupted, (4) both parties are corrupted." This is because the ideal functionality Figure 2 – and also Figure 10 from [35] – handles multiple users interacting with a server. 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 In this paragraph, we formally prove Theorem 1.

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, $ssid$) from Sim when a server sends OT-input to $\mathcal{F}_{\text{OT}}$. In the ideal world, Sim acts as $\mathcal{F}_{\text{OT}}$: This message is exactly formatted as a OT-SEND message from the functionality $\mathcal{F}_{\text{OT}}$. Further, Sim behaves exactly like $\mathcal{F}_{\text{OT}}$ in sending those messages. Concretely, on a message

$(\text{OT-SEND}, ssid, (X_i[0], X_i[1])_{i=1}^n)$, Sim stores the labels and informs the adversary that the labels were sent – but not which labels – by sending $(\text{OT-SEND}, ssid)$ to $\mathcal{A}$. This is exactly the behavior of $\mathcal{F}_{\text{OT}}$, as described in Figure 8. Additionally, Sim sends $(\text{OT-SEND}, ssid)$ messages to simulate an honest server, see line 29 in Figure 12. However, as this is done in reaction to a $(\text{SNDRCMPLT}, sid, ssid, \mathsf{S})$ message from $\mathcal{A}$, $\mathcal{E}$ expects to see this message. In the real world, the environment is only informed of an honest server sending a $(\text{OT-SEND}, ssid, (X_i[0], X_i[1])_{i=1}^n)$ message to $\mathcal{F}_{\text{OT}}$ by the $(\text{OT-SEND}, ssid)$ message. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real- or the ideal execution.

– $(\text{OT-RECEIVE}, ssid)$ from Sim when a user sends the choice bits to $\mathcal{F}_{\text{OT}}$ (simulated by Sim):
As above, we see that Sim behaves exactly like the original $\mathcal{F}_{\text{OT}}$. That means, on a message $(\text{OT-RECEIVE}, ssid, x)$, Sim stores the choice bits x and informs the adversary that the choice bits were received, but not which bits. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real or the ideal execution.

– $(\text{OT-SENT}, sid)$ from Sim when a corrupted server sent $(\text{OT-SEND}, sid, (X_i[0], X_i[1])_{i=1}^n)$ to $\mathcal{F}_{\text{OT}}$ before:
Again, Sim behaves like the real $\mathcal{F}_{\text{OT}}$ when creating those messages. Namely, upon receiving a message $(\text{OT-SENT}, sid)$ from the adversary, Sim ignores the message if $\langle sid, (X_i[0], X_i[1])_{i=1}^n \rangle$ or $\langle sid, x \rangle$ is not recorded; Otherwise Sim sends $(\text{OT-SENT}, sid)$ to $\hat{\mathsf{S}}$. Therefore, $\mathcal{E}$ cannot distinguish whether this message comes from the real or the ideal execution.

– $(\text{OT-RECEIVED}, sid, (X_i)_{i=1}^n)$ from Sim when a corrupted user sent $(\text{OT-RECEIVE}, ssid, x)$ to $\mathcal{F}_{\text{OT}}$ before:
These are the only messages on which Sim behaves not exactly as the real $\mathcal{F}_{\text{OT}}$. The messages are received by Sim when the environment "allows the delivery" of the OT-messages to the OT-receiver. If Sim recorded choice bits $x \neq \perp$, it means that a corrupted user sent $(\text{OT-RECEIVE}, ssid, x)$ before and Sim answers the query like $\mathcal{F}_{\text{OT}}$ 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 $\mathcal{F}_{\text{OT}}$ in the case when there is a OT-RECEIVED message $(\text{OT-RECEIVED}, ssid)$, and the recorded choice bits are $x = \perp$, see line 71 in Figure 14. The OT-RECEIVED message indicates that the input labels were sent via OT. The fact that $x = \perp$ is recorded for the choice bits means that Sim simulates a protocol execution for an honest user. If further a record $\langle sid, ssid, (F, K, d) \rangle$ or $\langle sid, ssid, (F, K, d), X[0^n], X[1^n] \rangle$ exists with the same $ssid$, all information for one OPRF execution was exchanged between server and user. 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 $\mathcal{F}_{\text{OT}}$. Thus, the simulator must also produce an output for honest users. The simulator retrieves the server identity S connected to sid. Sim sends $(\text{RCVCMPLT}, sid, ssid, \mathsf{U}, \mathsf{S})$ to $\mathcal{F}_{\text{OPRF}}$. The functionality $\mathcal{F}_{\text{OPRF}}$ 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 $\mathsf{tx}(sid) = 0$.
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 $\langle$GARBLE$, sid, ssid\rangle$ record was found. That record in turn is only created when an (EVAL$, sid, ssid, \mathsf{U}, \mathsf{S}$) message was received from $\mathcal{F}_{\text{OPRF}}$. 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 14, the ideal functionality will choose a random value 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 36.

Responses to Protocol Messages

- (GARBLE$, sid, ssid$) from $\mathcal{A}$ when a server receives this message, formatted as being sent from a user via $\mathcal{F}_{\text{AUTH}}$:
  In the ideal world, the message (GARBLE$, sid, ssid$) is sent by Sim as a reaction to an (EVAL$, sid, ssid, \mathsf{U}, \mathsf{S}$) message from $\mathcal{F}_{\text{OPRF}}$, because a real user would also start a protocol execution by requesting a garbled circuit from 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 (GARBLE$, sid, ssid$) message in the real world is is indistinguishable from this message created by Sim.
- $(sid, ssid, (F, K, d))$ from $\mathcal{A}$ when a user receives this message, formatted as being sent from a server via $\mathcal{F}_{\text{AUTH}}$:
  In the ideal world, this message is created by Sim, when Sim received a (SNDRCMPLT$, sid, ssid$) message. If the user of the subsession is corrupted, Sim also expects a (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 subsession with an honest user, Sim can simulate a (GARBLE$, sid, ssid$) message itself. The garbled circuit F and the decoding information d are calculated in the same way in both worlds, using $\mathsf{Gb}(1^\lambda, \mathcal{C})$. 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 worlds are distributed identically.

Responses of the Random Oracles:

- $H_1(p)$ 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 $x \in \{0,1\}^n$.

– $H_2(p, y)$ 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 Appendix C, 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 $\langle H_1, p, x\rangle$ found: The random oracle is programmed with a uniformly random value. In this case, Sim behaves like the real random oracle and sets the output to some random value.

Case 2: Records $\langle H_1, p, x\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exist, such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$: 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 $\mathcal{F}_{\text{OPRF}}$ for the correct output value by choosing an unused subsession id $ssid'$ and calling $(\textsc{Eval}, sid, ssid', \mathsf{S}, p)$ and $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$. If the ideal functionality does not answer, Sim aborts. Remember that $\mathcal{F}_{\text{OPRF}}$ 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 $\mathsf{tx}(sid) = 0$.
(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 $\mathcal{F}_{\text{OPRF}}$ just before sending $(\textsc{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$.
$H_2(p, y)$ is then programmed to the output ρ of $\mathcal{F}_{\text{OPRF}}$. This is, by the definition of $\mathcal{F}_{\text{OPRF}}$, a uniformly random value.

Case 3: There is a record $\langle H_1, p, x\rangle$ but no record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exists, such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$: In that case, Sim checks the keys of all corrupted parties $k_{\tilde{\mathsf{S}}}$. Note that in the ideal world, Sim knows those keys as it's Sim who delivers them (i.e., the whole internal state of the servers) to $\mathcal{E}$ on corruption. If there is such a corrupted server with key k_S such that $\mathcal{C}_{k_S}(x) = y$, 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.

In case $H_2(p, y)$ is already defined, Sim outputs that value again. This is the normal behavior of a random oracle.

Honest Serve Output (SndrCmplted, *sid*, *ssid*) from S at the end of a server's execution: In the real-world execution, the server outputs (SndrCmplted, *sid*, *ssid*) after successfully completing the OT to send the labels to the user. The simulator sends ok in line 58 of Figure 14 which is a reaction to (OT-Sent, *ssid*) message. Thus, the honest server will also produce this output in the ideal-world after successfully performing the proof.

Honest User Output (EvalOut, *sid*, *ssid*, ρ) from U as response to a message (Eval, *sid*, *ssid*, S, p):

In the real world, ρ is calculated as

$$\rho = H_2(p, \mathsf{De}(d, \mathsf{Ev}(F, X[x] \parallel K))),$$

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 $\mathcal{F}_{\text{OPRF}}$ if a fresh (Eval, *sid*, *ssid*, S, p) message was sent. Remember that $\mathcal{F}_{\text{OPRF}}$ 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 $\mathcal{F}_{\text{OPRF}}$ 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 $y = \mathcal{C}_k(H_1(p))$ holds for S's key k. We, therefore, have to compare the output of H_2 with the outputs of $\mathcal{F}_{\text{OPRF}}$. We distinguish the following cases in the simulation of H_2:

Case 1: There is no record $\langle H_1, p, x\rangle$ found: Sim only needs to program the random oracle, if p and y do occur in a protocol execution. More precisely, if $y = \mathcal{C}_k(H_1(p))$ holds for some server's key k. That is because in this case $\mathcal{F}_{\text{OPRF}}$ 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 $y = \mathcal{C}_k(H_1(p))$. 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 $k_1, \ldots, k_t$ be the keys used by the servers and let $n \in \Omega(\lambda)$ be the output length of $\mathcal{C}$. We assumed in the beginning that $\mathcal{C}_{k_i}(\cdot)$ is a permutation for every $i \in \{1, \ldots, t\}$. Thus, if we choose some uniformly random input $x \in \{0,1\}^n$, we get that $\mathcal{C}_{k_i}(x) \in \{0,1\}^n$ is uniformly random. If $H_1(p)$ is not queried yet, we have for every $i \in \{1, \ldots, t\}$ and every $y \in \{0,1\}^n$:

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

where the probability is taken over the random output of H_1. This follows from the fact that $\mathcal{C}_{k_i}(\cdot)$ 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:

$$\begin{aligned} \Pr[(p,y) \text{ becomes relevant}] &= \Pr\left[\bigvee_{i=1}^{t} (\mathcal{C}_{k_i}(H_1(p)) = y)\right] \\ &\leq \sum_{i=1}^{t} \Pr[\mathcal{C}_{k_i}(H_1(p)) = y] \\ &= t \Pr[\mathcal{C}_{k_1}(H_1(p)) = y] \\ &\overset{\text{Eq.2}}{\leq} \frac{t}{2^n}, \end{aligned} \tag{2}$$

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 $\langle H_1, p, x\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exist, such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$:
In this case, the value x 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 x 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 $\mathcal{F}_{\text{OPRF}}$ 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 $\mathcal{F}_{\text{OPRF}}$. 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 $\mathcal{F}_{\text{OPRF}}$. 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 $\langle H_1, p, x\rangle$ but no record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ exists, such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$:
In that case, the value x is the output of the random oracle H_1 on input p, but no honest server key maps x to $y = \mathcal{C}_k(x)$. 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 $\mathcal{C}_{k_{\hat{\mathsf{S}}}}(x) = y$ 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 $\mathcal{F}_{\text{OPRF}}$ and receives the answer $(\textsc{OfflineEval}, sid, \rho)$ from $\mathcal{F}_{\text{OPRF}}$. 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 the 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 15 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 4. When interacting with the simulator in Figures 11 to 15, for each server* S *the probability that a* $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ *message for* $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$ *is sent when the ideal functionality's ticket counter* $\mathsf{tx}(sid)$ *is 0, is negligible. That means, only with negligible probability* $\mathcal{F}_{\text{OPRF}}$ *ignores a* RcvCmplt *message because the ticket counter is 0.*

Proof. The ticket counter $\mathsf{tx}(sid)$ is only increased by $(\text{SndrCmplt}, sid, ssid)$ messages from S to $\mathcal{F}_{\text{OPRF}}$, i.e., by invocations of the server by $\mathcal{E}$. The counter is decreased by $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ messages for $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$ from Sim to $\mathcal{F}_{\text{OPRF}}$. The simulator from Figures 11 to 15 sends $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ messages in two cases. We will regard them separately:

Case 1: Sim received a query $H_2(p, y)$ and has records $\langle H_1, p, x\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0], X[1]\rangle$ such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$, i.e., the condition in line 85 in Figure 15 is true:

As Sim found the record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$, we can be sure that a $(\text{SndrCmplt}, sid, ssid)$ message was sent by $\mathcal{E}$ to Sim. This means the counter $\mathsf{tx}(sid)$ was increased at least once before the circuit was garbled. This holds, because Sim does only store the record $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ when it received a $(\text{SndrCmplt}, sid, ssid)$ message from $\mathcal{F}_{\text{OPRF}}$.
Next, we know that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$ 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 $\rho = T_{sid}(\mathsf{S}, x)$ value from $\mathcal{F}_{\text{OPRF}}$. We argue in Lemma 2 that another key $k' \neq k$ could lead to the same result y with at most negligible probability.
The $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ messages in line 74 of Figure 14 are only sent to produce an output of honest users. If the user is corrupted, that implies that there cannot be a message $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ produced by Sim in response to an $(\text{OT-Received}, ssid)$ message, in line 74 of Figure 14.
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 $\langle H_1, p, x\rangle$ and $\langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n]\rangle$ such that $\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$, happens at most with negligible probability.
In conclusion, sending $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ in line 89 of Figure 15 as a consequence of a $H_2(p, y)$ query will decrease the ideal functionality's counter

$\mathsf{tx}(sid)$ by one. Another $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ for the same S is sent in line 74 of Figure 14 at most with negligible probability. Querying the same tuple $H_2(p, y)$ again will not result in a second $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ message in line 89 of Figure 15, 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 once before with a $(\text{SndrCmplt}, sid, ssid)$ message to $\mathcal{F}_{\text{OPRF}}$.

Case 2: Sim received $(\text{OT-Received}, ssid)$ and a garbling (F, K, d) for a subsession $ssid$, where the recorded OT-request x is $\neq \perp$, i.e., the condition in line 71–73 of Figure 14 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 14 is true. We assume passive adversaries, which implies that a $(\text{SndrCmplt}, sid, ssid)$ message was already sent to $\mathcal{F}_{\text{OPRF}}$. Else, the server would not have created the garbling (F, K, d). This means, the counter $\mathsf{tx}(sid)$ is only decreased by one with a $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message in line 74 of Figure 14 if it is increased at least once before by a message $(\text{SndrCmplt}, sid, ssid)$ to $\mathcal{F}_{\text{OPRF}}$.

We argue why there cannot be another $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ message with $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$ for the same $sid, ssid$ and label S. There cannot be another $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message sent in line 74 of Figure 14 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 $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ once, at the moment when all n input labels are received by the user, i.e., when the condition in lines 71–73 of Figure 14 is true. If Sim receives further labels for the same $ssid$ it will not trigger a second $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message for this $ssid$. Remember that there are only two situations in which Sim sends $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, \mathsf{S})$ with $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}\}$. The first one is the situation where the message $(\text{OT-Received}, ssid)$ was 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. The recorded OT-request x is $\neq \perp$. Thus, the corresponding $(\text{OT-Receive}, ssid)$ message was simulated by Sim for an honest user. But the $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ messages in line 89 of Figure 15 are only sent if the key that "maps" $H_1(p)$ to y belongs to an honest server. Again, it follows from Lemma 3 that the $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ messages in line 89 of Figure 15 is sent at most with negligible probability.

In conclusion, the counter $\mathsf{tx}(sid)$ is only decreased by a $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ message sent in line line 74 of Figure 14 if there was a corresponding $(\text{SndrCmplt}, sid, ssid)$ message to $\mathcal{F}_{\text{OPRF}}$ before. The message $(\text{RcvCmplt}, sid, ssid, \mathsf{U}, \mathsf{S})$ is never sent twice for the same $ssid$ and there cannot be a corresponding $(\text{RcvCmplt}, sid, ssid', \mathcal{A}, \mathsf{S})$ message in line 89 of Figure 15 for the same S because of Lemma 3. □

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 a PRF. Let $t \in \mathbb{N}$ be polynomial in λ. For every fixed $x \in \{0,1\}^n$, and uniformly random and independently drawn keys $k_1, \ldots, k_t \in \{0,1\}^m$, there are, at most with negligible probability in λ, indices $i, j \in \{1, \ldots, t\}$ with $i \neq j$ such that $\mathsf{F}_{k_i}(x) = \mathsf{F}_{k_j}(x)$. This means, the simulator can assume in line 85 of Figure 15 that no other server has a key that maps the input $H_1(p)$ to the output y.*

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

$$\Pr[E_t = 1] = \Pr\left[\bigvee_{j=2}^{t} (y_j = \mathsf{F}_{k_1}(x))\right] \leq \frac{(t-1)}{2^l}, \tag{3}$$

where the probability is taken over the random choices of $y_2, \ldots, y_t$. Note, that $\mathsf{F}_{k_1}(x)$ is constant here. Assume, by way of contradiction, that the probability that the experiment E_1 outputs 1 with a noticeable probability. Then there is an index $N \in \{1, \ldots, t-1\}$ such that the difference $\Delta \coloneqq |\Pr[E_N = 1] - \Pr[E_{N+1} = 1]|$ is noticeable. We construct a distinguisher $\mathcal{D}$ for the PRF security experiment, 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 noticeable advantage Δ in the PRF experiment, which is a contradiction to F being a PRF. This concludes the hybrid argument.

In Equation (3), 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 $k_1, \ldots, k_t$. Hence, we have for all $x \in \{0,1\}^n$ that the probability that there are $i, j \in \{1, \ldots, t\}$ with $i \neq j$ such that $\mathsf{F}_{k_i}(x) = \mathsf{F}_{k_j}(x)$ is

$$\Pr\left[\bigvee_{i=1}^{t} \left(\bigvee_{j=1; j\neq i}^{t} \mathsf{F}_{k_i}(x) = \mathsf{F}_{k_j}(x) \right)\right] \leq \frac{t(t-1)}{2^l}, \tag{4}$$

which is negligible because in our case as $l \in \Omega(\lambda)$. □

Lemma 3. *Let the garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ *have privacy, as defined in Definition 4. Let* $\mathcal{C}$ *be the boolean circuit of a PRF, as defined in Definition 1. Suppose the environment* $\mathcal{E}$ *initiates an OPRF execution between an honest server and an honest user with input* p*. The environment* $\mathcal{E}$ *can at most with negligible probability send a request* $H_2(p, y) \in \{0,1\}^n$ *such that* $\mathcal{C}_k(H_1(p)) = y$*, where* $k \in \{0,1\}^m$ *is the key of the honest server.*

Proof. Without loss of generality, we can assume that $\mathcal{E}$ requested $x = H_1(p)$ for the user input $p \in \{0,1\}^*$. Further, we assume that $\mathcal{E}$ 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, $\mathcal{E}$ received no labels for the user input x, as Sim simulated the OT for an honest user.

Assume, by way of contradiction, that $\mathcal{E}$ calculates an output $y \in \{0,1\}^n$ such that
$\mathsf{De}(d, \mathsf{Ev}(F, X[x] \,\|\, K)) = y$ with noticeable probability P. First, we construct an adversary $\mathcal{B}$ that plays the privacy experiment as in Figure 6 and communicates with $\mathcal{E}$ as if $\mathcal{B}$ was the simulator. As we assume that the garbling scheme has privacy, we will get the existence of a simulator $\mathsf{Sim}_{\text{PRF}}$ for the privacy experiment. We will use this simulator $\mathsf{Sim}_{\text{PRF}}$ and the adversary $\mathcal{B}$ to construct a second adversary $\mathcal{B}_{\text{PRF}}$ that will have a noticeable success probability in distinguishing the PRF $\mathcal{C}$ from a truly random function, which is a contradiction to our assumption that $\mathcal{C}$ satisfies Definition 1 of a PRF.

$\mathcal{B}$ plays the UC-security experiment with $\mathcal{E}$. Let $t \in \mathbb{N}$ be the number of subsessions that $\mathcal{E}$ invokes between an honest server and an honest user. The adversary $\mathcal{B}$ initially chooses an index $i^* \in \{1, \ldots, t\}$ uniformly at random. $\mathcal{B}$ behaves like our normal simulator from Figures 11 to 15, except when $\mathcal{E}$ 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 $(\textsc{SndrCmplt}, sid, ssid, \mathsf{S})$ message from $\mathcal{F}_{\text{OPRF}}$, $\mathcal{B}$ 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 $\mathsf{C}_{\text{privacy}}$. 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 $(x', k, \mathcal{C})$ to $\mathsf{C}_{\text{privacy}}$. The privacy challenger chooses $b \in \{0,1\}$ uniformly at random. If $b = 1$, it calculates $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{C})$ and $(\tilde{X}, K) = \mathsf{En}(e, x' \,\|\, k)$. If $b = 0$, it calculates $y' = \mathsf{ev}(\mathcal{C}, x' \,\|\, k) = \mathcal{C}_k(x')$. Next, $\mathsf{C}_{\text{privacy}}$ runs the simulator $\mathsf{Sim}_{\text{PRF}}$ on input y'. The simulator $\mathsf{Sim}_{\text{PRF}}$ 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 $\mathcal{E}$, formatted as if U sent it to S via $\mathcal{F}_{\text{AUTH}}$. Note that $\tilde{X}$ is *not* sent to $\mathcal{E}$ as our actual OPRF simulator from Figures 11 to 15 would also not do that. Finally, $\mathcal{B}$ checks for ever H_2 query (p, y^*) from $\mathcal{E}$, if $y^* = \mathcal{C}_k(H_1(p))$ holds. Only if that is the case, $\mathcal{B}$ outputs 1, else it outputs 0.

In the case where the challenger $\mathsf{C}_{\text{privacy}}$ chose $b = 1$, the view of $\mathcal{E}$ 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 $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{C})$. The calculation of those values is completely independent of the value x'. The encoded key is calculated as $(\tilde{X}, K) = \mathsf{En}(e, x' \,\|\, k)$, 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 $\mathcal{E}$ succeeds in calculating y^* such that $y^* = \mathcal{C}_k(H_1(p))$ 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 $\mathcal{E}$ 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 $\mathcal{E}$. The adversary $\mathcal{B}_{\mathrm{PRF}}$ chooses an index $i^* \in \{1, \ldots, t\}$ 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 $(\textsc{Eval}, sid, ssid, \mathsf{U}, \mathsf{S})$ message and a $(\textsc{SndrCmplt}, sid, ssid, \mathsf{S})$ 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 $\mathsf{C}_{\mathrm{PRF}}$. The challenger $\mathsf{C}_{\mathrm{PRF}}$ chooses a bit $b' \in \{0,1\}$ uniformly at random. If $b' = 1$, the challenger calculates $\hat{y} = \mathcal{C}_{k'}(\hat{x})$, for some uniformly random $k' \in \{0,1\}^m$. If $b' = 0$, the challenger $\mathsf{C}_{\mathrm{PRF}}$ sets $\hat{y} = \mathsf{RF}(\hat{x})$ where $\mathsf{RF} \in \{f : \{0,1\}^n \to \{0,1\}^n\}$ is chosen uniformly at random. $\mathsf{C}_{\mathrm{PRF}}$ 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 $\mathcal{E}$ as if the honest server sent (F, K, d) to the honest user via $\mathcal{F}_{\mathrm{AUTH}}$. The environment $\mathcal{E}$ answers with a value $\bar{y}$. Now, $\mathcal{B}_{\mathrm{PRF}}$ checks for every H_2 query $(p, \bar{y})$ if $\bar{y} = \mathcal{C}_k(H_1(p))$ holds. Only if that is true, $\mathcal{B}_{\mathrm{PRF}}$ outputs 1, else it outputs 0.

Suppose that $\mathcal{B}_{\mathrm{PRF}}$ chose the correct index i^*, i.e., the subsession in which $\mathcal{E}$ 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 $\mathcal{E}$ has noticeable probability P' to send a query $(p, \bar{y})$ such that $\bar{y} = \mathcal{C}_k(H_1(p))$. 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 $\mathcal{C}_k(H_1(p))$. In that case, the input (F, K, d) gives $\mathcal{E}$ information-theoretically no advantage in guessing $\mathcal{C}_k(H_1(p))$. Consequently, $\mathcal{E}$ outputs $(p, \bar{y})$ such that $\bar{y} = \mathcal{C}_k(H_1(p))$ 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 exist, which is a contradiction to the assumed privacy of the garbling scheme. Thus, the assumed environment $\mathcal{E}$ cannot exist. □

D Omitted Proofs from Section 5

In the following, we give a proof for Claim 1.

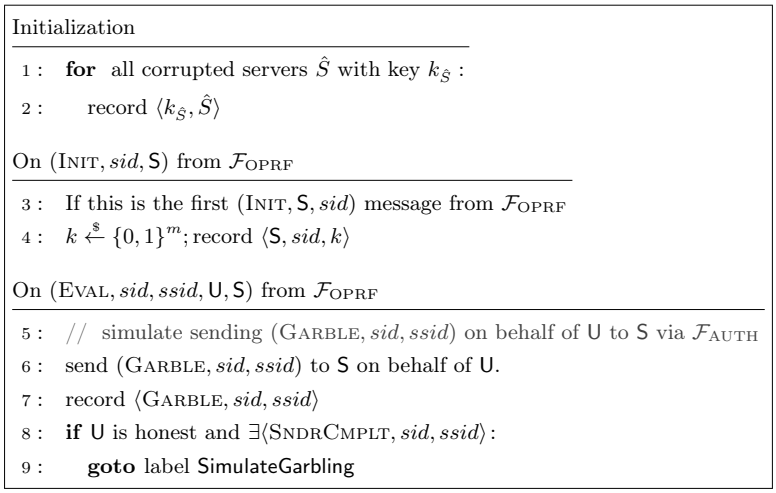

Fig. 11: The simulator Sim part I. Initialization and simulation of receiving a Eval message.

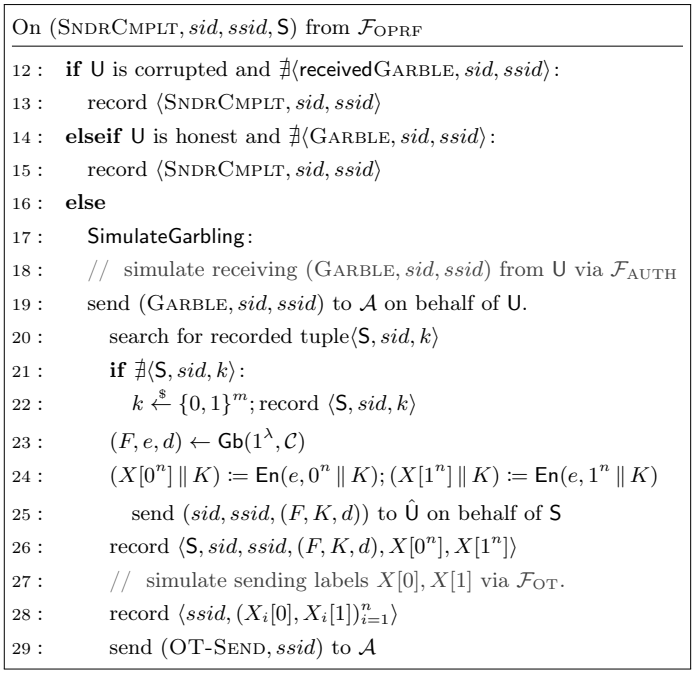

Fig. 12: The simulator Sim part II. Simulation of receiving a SndrCmplt message.

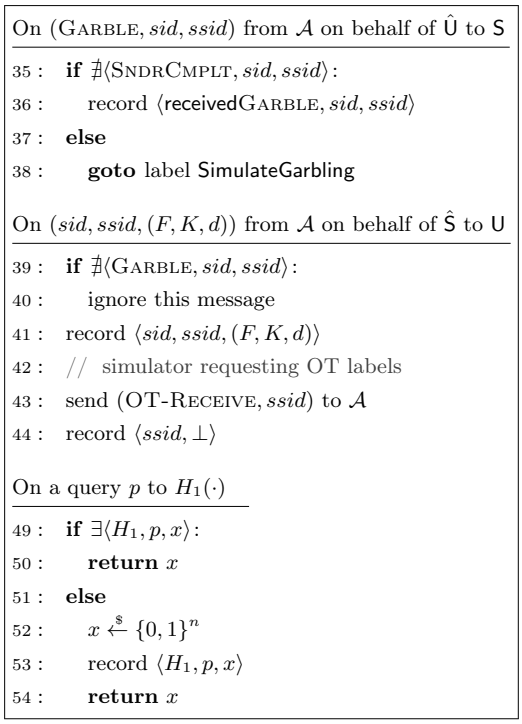

Fig. 13: The simulator Sim part III. Simulation of protocol messages and the first random oracle H_1.

On (OT-Send, $ssid$, $(X_i[0], X_i[1])_{i=1}^n$) from $\mathcal{A}$ to $\mathcal{F}_{\text{OT}}$ on behalf of $\hat{\mathsf{S}}$

record $\langle \hat{\mathsf{S}}, ssid, (X_i[0], X_i[1])_{i=1}^n \rangle$
send (OT-Send, $ssid$) to $\mathcal{A}$.
ignore further (OT-Send, $ssid, \dots$) messages

On (OT-Sent, $ssid$) from $\mathcal{A}$ to $\mathcal{F}_{\text{OT}}$

if S is honest, send ok to $\mathcal{F}_{\text{OPRF}}$
elseif $\nexists \langle \hat{\mathsf{S}}, ssid, (X_i[0], X_i[1])_{i=1}^n \rangle$:
  ignore this message
else
  send (OT-Sent, $ssid$) to $\hat{\mathsf{S}}$
ignore further (OT-Sent, $ssid$) messages

On (OT-Receive, $ssid, x$) from $\mathcal{A}$ to $\mathcal{F}_{\text{OT}}$ on behalf of $\hat{\mathsf{U}}$

record $\langle ssid, x \rangle$
send (OT-Receive, $ssid$) to $\mathcal{A}$
ignore further (OT-Receive, $ssid$) messages

On (OT-Received, $ssid$) from $\mathcal{A}$ to $\mathcal{F}_{\text{OT}}$

if $\nexists \langle \mathsf{S}, ssid, (X_i[0], X_i[1])_{i=1}^n \rangle$ or $\nexists \langle ssid, x \rangle$:
  ignore this message
elseif $x \neq \perp$
  send (OT-Received, $ssid$, $(X_i[x_i])_{i=1}^n$) to $\hat{\mathsf{U}}$
else
  if $(\exists \langle sid, ssid, (F, K, d) \rangle$
    or $\exists \langle \mathsf{S}, sid, ssid, (F, K, d), X[0^n], X[1^n] \rangle)$:
  send (RcvCmplt, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathcal{F}_{\text{OPRF}}$

Fig. 14: The simulator Sim part IV. Simulation of $\mathcal{F}_{\text{OT}}$.

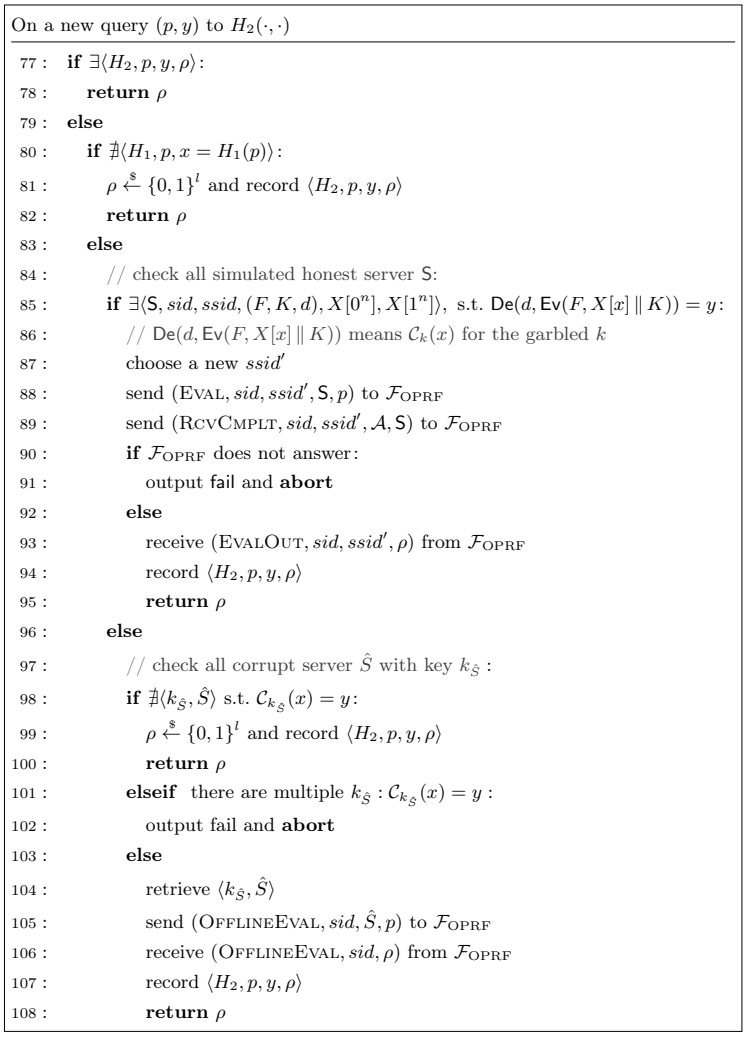

Fig. 15: The simulator Sim part V. Simulation of the second random oracle H_2.

Proof. A simulator for this works straightforwardly. The simulator just relays all messages to $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$. When it receives a COMPROMISE message, it also relays this message to $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ and receives the key k, chosen by the functionality. With this, the simulator can answer all OFFLINEEVAL queries consistently by computing $\mathsf{F}_k(p)$ himself. The environment never sees k. However, by the definition of $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ the key k is chosen uniformly at random. Thus, if the environment can distinguish the "real-world" experiment with $\mathcal{F}_{\text{OPRF}}$ and an adversary from the "ideal-world" experiment with $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ and the simulator, we can use the environment $\mathcal{E}$ to construct an adversary $\mathcal{B}$ in the PRF security game, see Definition 1. In the game, $\mathcal{B}$ gets access to an oracle that answers either with truly random values or values from a PRF F_k for some uniformly random k in the key space of the PRF. $\mathcal{B}$ internally executes $\mathcal{E}$ and interacts with $\mathcal{E}$ in the UC-experiment. $\mathcal{B}$ plays the role of the ideal functionality. Whenever the functionality would send $(\text{EVALOUT}, sid, \rho)$ (or $(\text{OFFLINEEVAL}, sid, \rho)$) to a party as reaction to a $(\text{EVAL}, sid, ssid, \mathsf{S}, p)$ (or $(\text{OFFLINEEVAL}, sid, i, p)$) message, $\mathcal{B}$ sends p to the PRF oracle and sets ρ to whatever the oracle returns. If $\mathcal{E}$ halts and outputs a bit b, the adversary $\mathcal{B}$ outputs the same bit b.

If the PRF oracle answers with truly random values, the view of $\mathcal{E}$ is distributed as in the "real-world" experiment with $\mathcal{F}_{\text{OPRF}}$. If the PRF oracle answers with values from a PRF, the view of $\mathcal{E}$ is distributed as in the "ideal-world" experiment with $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ and the simulator. Hence, $\mathcal{B}$ has the same success probability as $\mathcal{E}$, which is by assumption negligible. □

Next, we give a proof for Claim 2.

Proof. If $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ UC-realizes $\mathcal{F}_{\text{OPRF}}$ then we have for the dummy adversary $\mathcal{D}$ interacting with $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$ a simulator Sim interacting with $\mathcal{F}_{\text{OPRF}}$ such that no environment $\mathcal{E}$ can tell the two executions apart with noticeable probability. We describe in the following why such a simulator cannot exist. Suppose that $\mathcal{E}$ initializes a session between a server S and a user U. Let's denote by $n, m, l \in \mathbb{N}$ the input length, the key length, and the output length of the PRF, respectively. Now $\mathcal{E}$ chooses arbitrary input values $p_1, \ldots, p_N \in \{0,1\}^n$ for some polynomial $N := N(\lambda)$, where $N > (m+\lambda)/l$. The environment proceeds by sending messages $(\text{EVAL}, sid, ssid_1, \mathsf{S}, p_1), \ldots, (\text{EVAL}, sid, ssid_N, \mathsf{S}, p_N)$ to the user, and respectively $(\text{SNDRCMPLT}, sid, ssid_j)$ and $(\text{RCVCMPLT}, sid, ssid_j, \mathsf{U}, \mathsf{S})$ messages to the server and the adversary. In the "real-world" experiment, U will receive a $(\text{EVALOUT}, sid, \rho_j)$ message for each input p_j and will output the received value to $\mathcal{E}$. Therefore, every valid simulator must ensure that the user also receives $(\text{EVALOUT}, sid, \rho_j)$ messages in the "ideal-world" experiment. In the "ideal-world" the values ρ_j will be chosen uniformly at random in $\{0,1\}^l$ by the ideal functionality. Note that the simulator may change from which table $T_{sid}(i, p_j)$ the values will be sampled but every entry in the table is uniformly random. That is because a user will only receive output if Sim sends $(\text{RCVCMPLT}, sid, ssid_j, \mathsf{P}, i)$ to $\mathcal{F}_{\text{OPRF}}$, where i indicates the random table $T_{sid}(i, \cdot)$ from which the output is taken.

Now, $\mathcal{E}$ sends a $(\text{COMPROMISE}, sid, \mathsf{S})$ message to the adversary. In the "real-world" execution with $\mathcal{F}_{\text{OPRF}}^{\mathsf{F}}$, the environment will receive a key $k \in \{0,1\}^m$

such that all output values $\rho_1, \ldots, \rho_N$ are PRF outputs with respect to k, i.e., $\rho_j = \mathsf{F}_k(p_j)$. But that means that a simulator now must come up with a key k that explains the uniformly random values $\rho_1, \ldots, \rho_N$. For a fixed PRF key k, the function $\mathsf{F}_k(\cdot)$ is deterministic. For each input $p_j \in \{0,1\}^n$ we let $y_j \coloneqq \mathsf{F}_k(p_j)$. Now, we have for a uniformly random output $\rho_j \in \{0,1\}^l$ that $y_j = \rho_j$ at most with probability 2^{-l}, over the randomness of ρ_j. Thus, the probability that the key k fits to all N inputs is at most 2^{-Nl}. There are 2^m different keys. So the probability that at least one of the keys maps all N inputs to the corresponding ρ_i value is $2^{m-Nl} \leq 2^{-\lambda}$, which is negligible. □

The next proof shows Claim 3.

Proof. Assume, by way of contradiction, π to be a protocol with reproducible output that UC-realizes $\mathcal{F}_{\text{OPRF}}$ in the NPROM. We have to show that for every simulator $\mathsf{Sim}^{\mathcal{O}}$ there is an environment $\mathcal{E}^{\mathcal{O}}$ that can distinguish $\text{EXEC}_{\pi,\mathcal{D}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}}$ and $\text{IDEAL}_{\mathcal{F}^{\mathcal{O}}_{\text{OPRF}},\mathsf{Sim}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}}$ with noticeable probability, where $\mathcal{D}^{\mathcal{O}}$ denotes the dummy adversary. Remember that we are in the NPROM and thus all parties—even the environment—get access to a random oracle $\mathcal{O}$. We write $m' = \pi(\mathsf{P}, m, \mathit{state}, r_{\mathsf{P}})$ to denote that a P following protocol π will send message m' as next message, when P received the message m while being in state *state* and having random-tape r_{P}. The intuition for the proof is that the environment is going to pre-compute the output of the protocol execution "in its head" and will compare this output to the actual execution. The environment $\mathcal{E}$ starts and chooses randomness $r_{\mathsf{U}}, r_{\mathsf{S}} \in \{0,1\}^\lambda$ uniformly at random and some (arbitrary) input value p. It then internally executes the protocol π between a user $\mathsf{U}(p; r_{\mathsf{U}})$ and a server $\mathsf{S}(r_{\mathsf{S}})$. Whenever one of the parties makes a query to the random oracle, $\mathcal{E}$ forwards this message to $\mathcal{O}$ and also forwards the response of $\mathcal{O}$. At the end of the execution, U will output a value y. We denote the internal state of the server at the end of the execution as k. Note that $\mathcal{E}$ knows this state k as S is executed internally by $\mathcal{E}$. Next, $\mathcal{E}$ invokes a user U' and a corrupted server S' externally, meaning in the actual UC-experiment. The environment $\mathcal{E}$ instructs the user U' to execute the protocol π with server S'. On every message m to S', $\mathcal{E}$ calculates $m' = \pi(\mathsf{P}, m, k, r_{\mathsf{S}'})$, and instructs the adversary $\mathcal{D}$ to send m' to U' on behalf of S'. At the end of the execution, U' outputs a value y'. The environment outputs 1 iff $y \neq y'$.

In the "real-world" execution $\text{EXEC}_{\pi,\mathcal{D}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}}$, the users U and U' have the same view as two users with input p who interact with a server S that is in state k after the first execution. As π is a protocol with reproducible output, we have that $y = y'$ with overwhelming probability.

In the "ideal-world" execution $\text{IDEAL}_{\mathcal{F}^{\mathcal{O}}_{\text{OPRF}},\mathsf{Sim}^{\mathcal{O}},\mathcal{E}^{\mathcal{O}}}$ the simulator Sim must produce an output for the (honest) user U', as U' also had some output in the real-world execution. This means, Sim must send a $(\text{RcvCmplt}, \mathit{sid}, \mathit{ssid}, \mathsf{P}, i)$ message to $\mathcal{F}_{\text{OPRF}}$. The user's output y' will be uniformly random in $\{0,1\}^l$, as it comes directly from the ideal functionality. In particular, it will be independent of the output y, which $\mathcal{E}$ computed before "in its head". That means that $y = y'$ at most with probability 2^{-l}. Consequently, any protocol that UC-realizes $\mathcal{F}_{\text{OPRF}}$

in the ROM (without using additional hybrid functionalities) must program the Random Oracle in the security proof. □

Finally, we show Claim 4.

Proof. Let π be the protocol as in Figure 9, where the server garbles the circuit of a PRF F and chooses a uniformly random key k as input to the circuit. The user gives its input p as choice bits to the OT protocol. Finally, the user evaluates the circuit and outputs the output of the circuit as OPRF output. Again, we have to give an environment that distinguishes the real-world execution from the ideal-world execution.

As in the proof above, our environment will pre-calculate the output of the garbled circuit as $\mathsf{F}_k(p) =: \rho$. The environment will initialize an execution of the protocol between an honest user with input p and a corrupted server with key k. The environment will then observe the output of this execution. In the real-world, the execution will output ρ, by the correctness of the garbling scheme. In the ideal-world, the simulator must produce an output ρ' for the user by sending a $(\text{RcvCmplt}, sid, ssid, \mathsf{P}, i)$ message to $\mathcal{F}_{\text{OPRF}}$. Again, Sim can only influence the output by choosing a different value $i' \neq i$. But regardless of which i' it is choosing, the resulting output for the user will be uniformly random. Also, note that manipulating the responses of $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{RO}}$ will not help Sim. The OT-messages $(\text{OT-Send}, ssid)$ and $(\text{OT-Sent}, ssid)$ messages that Sim produces in the execution do not affect the user's output. The programming of the random oracle might indeed be used to manipulate the output of $\mathsf{Ev}(F, K, X)$ to be ρ. But the environment does not do its pre-computation with the garbled version of the circuit (F, K, X) but with the "plain" version $\mathsf{F}_k(p)$. Consequently, it means that still $\rho \neq \rho'$ with high probability. □

Note that the above poof does not hold for 2HashDH [35] because the environment cannot pre-compute the OPRF output $H'(p, H(p)^k)$ without the simulator being able to tamper with the RO queries.