SoK: Oblivious Pseudorandom Functions

Sílvia Casacuberta
Harvard University
scasacubertapuig@college.harvard.edu

Julia Hesse
IBM Research Europe - Zurich
jhs@zurich.ibm.com

Anja Lehmann
Hasso Plattner Institute &
University of Potsdam
anja.lehmann@hpi.de

***Abstract*—In recent years, oblivious pseudorandom functions (OPRFs) have become a ubiquitous primitive used in cryptographic protocols and privacy-preserving technologies. The growing interest in OPRFs, both theoretical and applied, has produced a vast number of different constructions and functionality variations. In this paper, we provide a systematic overview of how to build and use OPRFs. We first categorize existing OPRFs into essentially four families based on their underlying PRF (Naor-Reingold, Dodis-Yampolskiy, Hashed Diffie-Hellman, and generic constructions). This categorization allows us to give a unified presentation of all oblivious evaluation methods in the literature, and to understand which properties OPRFs can (or cannot) have. We further demonstrate the theoretical and practical power of OPRFs by visualizing them in the landscape of cryptographic primitives, and by providing a comprehensive overview of how OPRFs are leveraged for improving the privacy of internet users.**

Our work systematizes 15 years of research on OPRFs and provides inspiration for new OPRF constructions and applications thereof.

1. Introduction

In the 80's, Goldreich et al. [1] demonstrated that it is indeed possible to efficiently construct deterministic functions whose output looks like it is chosen at random. Pseudorandom functions (PRFs) have ever since been a core cryptographic primitive with innumerable applications and instantiations. Two decades later, Naor and Reingold [2], [3] noticed that their number-theoretic PRF allows for an *interactive* and *oblivious* evaluation, where a "client" with input x obtains $\mathsf{PRF}_k(x)$ for a function $\mathsf{PRF}_k(\cdot)$ that is contributed by a "server". Neither does the client learn the function (i.e., its key k), nor does the server learn x or $\mathsf{PRF}_k(x)$. Freedman et al. [4] later called such two-party protocol an *oblivious pseudorandom function* (OPRF) and gave first formal definitions and two OPRFs based on the Naor-Reingold PRF [3].

The destiny of OPRFs: Protecting our privacy. Although Freedman et al. [4] directly demonstrated how OPRFs were useful for obliviously searching a database, OPRFs did not immediately receive a lot of attention. A few years later, Hazay and Lindell [5] discovered their close relationship to private set intersection (PSI), underlining the role OPRFs can play in our everyday lifes: their obliviousness allows to construct protocols that protect private data, such as our passwords, our search inputs, our identities, our digital footprints. And indeed, in the past decade, OPRFs have shown to be a central primitive for building oblivious keyword search (KS) [4], [6], private set intersection (PSI) [5], [6], [7], [8], password-protected secret sharing (PPSS/TPASS) [9], [10], [11], [12], [13], private information retrieval (PIR) [4], password-authenticated key exchange (PAKE) [9], [14], single sign-on (SSO) with privacy [12], cloud key management [15], de-duplication systems [16], secure pattern matching [5], [17], and "untraceable" contact tracing [18].

Naturally, adapting OPRFs to all these applications required the introduction of several additional properties: OPRFs that are verifiable [9], [13], [19], [20], [21], have correlated [13] or committed inputs [7], [9] or outputs [22], have updatable keys [12], [19], partially reveal their input to the server [12], [13], [19], work in the threshold or distributed server setting [11], [12], [13], or allow for batching techniques [6].

Where are we now? More than 15 years after their introduction, OPRFs have become an established and well researched part of cryptography. Being at the core of privacy-preserving technologies, OPRFs enjoy a growing interest and, consequently, efforts arise towards standardization [23], [24], [25]. However, the downside of the high activity is also visible from the literature. There is an increasing list of properties, partly repeated under ambiguous or different names. Some properties seem to partly or fully imply, or even contradict, one another. By now, many variants of initial OPRFs have evolved in the literature, and keeping track of which already exist has become increasingly hard. On top of that, there exist various notions of security for OPRFs (e.g., weak, strong, game-based, securely computable, or composable). Lastly, from a more theoretical point of view, it is unclear where OPRFs stand in the landscape of cryptographic primitives, with individual relations to other primitives proven often only implicitly in the numerous literature.

Our contributions and outline of the paper. In this paper we provide the first comprehensive overview of oblivious pseudorandom functions. We highlight both theoretical aspects of OPRFs as a cryptographic object, as well as their practical role as a building block for many applications. In Section 2 we categorize OPRFs from the literature according to their underlying PRF, and explain all

This work was partly supported by the EU Horizon 2020 research and innovation programme under grant agreement No 786725 OLYMPUS. The second author was partly supported by SNF AMBIZIONE grant "Your Biometrics, please! Cryptographic Protocols for Human Authentication and the IoT".

known methods for obliviously evaluating them. Section 3 describes functional OPRF properties, such as sharing or update of the key, verifiability, or partial revelation of inputs. We explain which OPRF constructions can (or cannot) achieve which properties. Section 4 explains in which models one can formulate OPRF security. In Section 5, we consider practical applications of OPRFs, and observe that they can be divided into essentially two categories: (1) retrieval of cryptographic material, and (2) enforcing interaction when computing hash values; e.g., to allow for rate limiting. We give examples, blueprint protocols and pointers to papers and real-world deployments for the above ways on how to leverage OPRFs. We conclude by suggesting several research directions that we believe will improve adaptation of OPRFs in the future in Section 6. The appendix provides a gentle introduction to OPRFs for non-cryptographers (Appendix A). Appendix B discusses the theoretical impact of OPRFs and visualizes them in the landscape of cryptography. This constitutes the first comprehensive source of information on how OPRFs can be constructed from other primitives. This section also includes a classification of OPRFs that have been developed case-tailored to PSI (Appendix B.4).

2. Constructions

To gain an overview of the many OPRF constructions in the literature, we first observe that we can sort them into mainly four categories. In Table 1 we list all practically-relevant OPRFs from the literature based on their underlying PRF and high-level method of oblivious evaluation. In this survey we focus on OPRFs that are more efficient than simply computing the evaluation circuit with generic multi-party computation (MPC) techniques. Before explaining each of these categories in more detail, we give a first formal definition of an OPRF. We recommend readers unfamiliar with PRFs to first read the non-technical introduction to OPRFs in Appendix A.

Using MPC terminology, an OPRF protocol for the PRF $f_k(\cdot)$ is a secure computation of the functionality $(k, x) \to (\perp, f_k(x))$. More formally, we obtain the following first definition.

Definition 1 (Oblivious pseudorandom function, [4]). A two-party protocol π between a client and a server is an *oblivious pseudorandom function (OPRF)* if there exists some PRF family f_k, such that π privately realizes the following functionality:

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

Freedman et al. [4] call the above functionality a *strong* OPRF, as opposed to a *weak* (also called *relaxed*) OPRF [4, Def. 6] which does not prevent the client from learning partial information about the key.

2.1. Naor-Reingold PRF

Let p, q be primes such that $q \mid p-1$, $n \in \mathbb{N}$, $k := (a_0, \dots, a_n) \leftarrow_{\text{R}} \mathbb{Z}_q^{n+1}$ and $g \in \mathbb{Z}_p$ is an element of order q. The Naor-Reingold (NR) PRF [2], [3] with key k on input $x := x_1 x_2 \dots x_n \in \{0,1\}^n$ is defined as follows:

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

PRF	OPRFs	Method
Naor-Reingold	[4], [5], [21], [26]	Obl. Transfer
	[9], [27]	Hom. encryption
HashedDH	[9], [10], [11], [14] [12], [13], [28], [29]	Blinded exp.
Dodis-Yampolskiy	[7], [16], [22], [30]	Hom. encryption
	[31]	Blinded exp.
Any	[6], [32], [33] [18], [34], [35], [36]	Obl. Transfer

TABLE 1: Classification of OPRF protocols based on their underlying PRF and method of oblivious evaluation.

i.e., raise g^{a_0} to a_i if and only if the i-th bit of x is 1. This function's outputs on chosen inputs are indistinguishable from randomly sampled elements from the group $\langle g \rangle$, and hence are *pseudorandom*, under the decisional Diffie-Hellman assumption (DDH) [2]. The DDH assumption is a standard discrete-log type assumption and is believed to hold in many groups. Naor and Pinkas [37], [38] later demonstrated that pseudorandomness also holds if the group is of prime order, allowing for more efficient instantiations.

A protocol for evaluating the prime-order variant of f^{NR} obliviously, and at the same time the first formal OPRF protocol in the literature, was given by Freeman et al. [4, §5.1]. Their OPRF leverages oblivious transfer (OT; see Appendix B.1 if unfamiliar) and is depicted in Figure 1. On a high level, the server S chooses fresh randomness $\vec{r}$ to blind the key $\vec{a}$, and for each input bit the client C receives either only the blinding or the blinded key part via oblivious transfer.

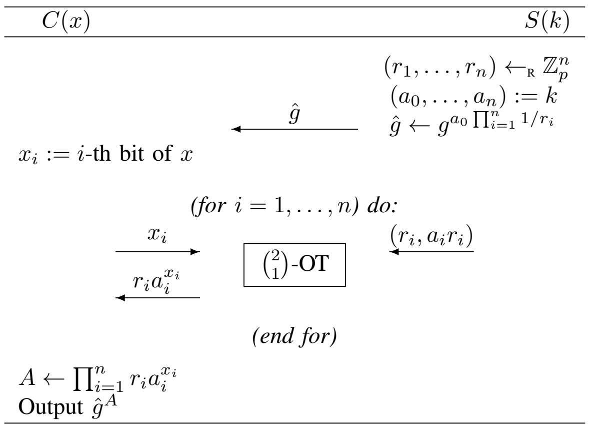
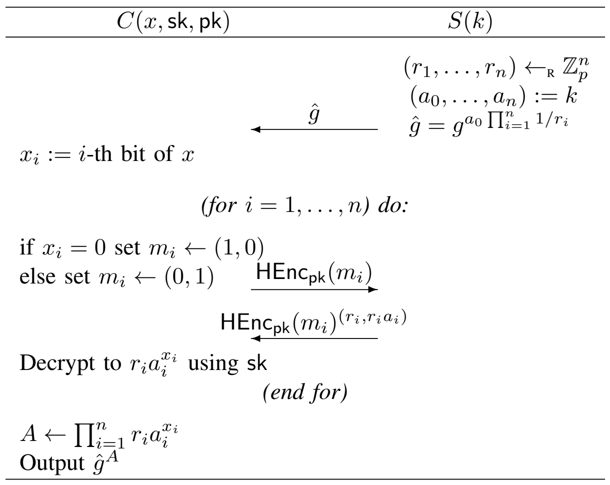

$C(x)$
$S(k)$
$(r_1, \dots, r_n) \leftarrow_{\text{R}} \mathbb{Z}_p^n$
$(a_0, \dots, a_n) := k$
$\hat{g} \leftarrow g^{a_0 \prod_{i=1}^{n} 1/r_i}$
$\hat{g}$
$x_i := i$-th bit of x
(for $i = 1, \dots, n$) do:
x_i
$(r_i, a_i r_i)$
$\binom{2}{1}$-OT
$r_i a_i^{x_i}$
(end for)
$A \leftarrow \prod_{i=1}^{n} r_i a_i^{x_i}$
Output $\hat{g}^A$

Figure 1: Oblivious evaluation of the NR PRF using OT.

Correctness of the protocol follows from cancellation of the blinding values r_i, and the protocol is secure (as in Def. 1) if DDH is hard in the group, w.r.t a semi-honest server. The final cost is that of n $\binom{2}{1}$-OTs (which can be performed efficiently with batched OT techniques [39]) and one exponentiation. The protocol was later shown secure against malicious servers [5], still under DDH, by using a maliciously secure OT variant.

Jarecki et al. [9] provided a *universally composable* (UC) [40] OPRF based on f^{NR} in the Common Reference String (CRS) model. In their construction, they allow the client to additionally receive a "public key" version of k (which can be thought of it as, e.g., g^k) along with $f_k^{\mathsf{NR}}(x)$, and verify correct computation w.r.t this public key. Such an OPRF is a versatile building block, since it allows to enforce re-use of the server's key. Note that in Definition 1, a client is only guaranteed to receive a correctly computed value w.r.t whatever input the server provided, but cannot find out whether a server uses the *same or a different* key in another protocol run [41]. We will dive into all these aspects of formalizing OPRFs in more sophisticated ways than Definition 1 in Section 4.



$C(x, \mathsf{sk}, \mathsf{pk})$
$S(k)$
$(r_1, \ldots, r_n) \leftarrow_R \mathbb{Z}_p^n$
$(a_0, \ldots, a_n) := k$
$\hat{g} = g^{a_0 \prod_{i=1}^{n} 1/r_i}$
$\hat{g}$
$x_i :=$ i-th bit of x
(for $i = 1, \ldots, n$) do:
if $x_i = 0$ set $m_i \leftarrow (1, 0)$
else set $m_i \leftarrow (0, 1)$
$\mathsf{HEnc}_{\mathsf{pk}}(m_i)$
$\mathsf{HEnc}_{\mathsf{pk}}(m_i)^{(r_i, r_i a_i)}$
Decrypt to $r_i a_i^{x_i}$ using sk
(end for)
$A \leftarrow \prod_{i=1}^{n} r_i a_i^{x_i}$
Output $\hat{g}^A$

Figure 2: Oblivious evaluation of the NR PRF using homomorphic encryption HEnc on message tuples m_i, omitting proofs of honest behavior.

We depict a blueprint for oblivious evaluation of the NR PRF using homomorphic encryption in Figure 2. An encryption scheme HEnc is homomorphic if (using the multiplicative notation) $\mathsf{HEnc}_{\mathsf{pk}}(\vec{a})^{\vec{b}} := \mathsf{HEnc}_{\mathsf{pk}}(a_1)^{b_1} \cdot \mathsf{HEnc}_{\mathsf{pk}}(a_2)^{b_2} = \mathsf{HEnc}_{\mathsf{pk}}(\langle \vec{a}, \vec{b} \rangle)$, where $\langle \cdot, \cdot \rangle$ denotes the inner product. The idea on how to leverage such an encryption scheme to build an OPRF is similar to the NR-OPRF from OT of Figure 1, except that this time one value chosen out of $(r_i, r_i a_i)$ is transmitted from S to C in encrypted form. Usage of homomorphic encryption allows to equip the blueprint in Figure 2 with verifiability (the server performs ZK proofs of honest behavior w.r.t public key $g^{a_0}, \ldots, g^{a_n}$, requiring also additional commitments) and security against malicious parties (efficiently implementable with several parallel Σ-protocols [9]).

Finally, we mention a variant of the NR PRF with compact key $k := (k_0, k_1) \leftarrow_R \mathbb{Z}_q^2$ for inputs $x < D$ that are polynomial in the security parameter. Then the function $f^{\mathsf{NR}}_{\mathsf{poly}}(x) := g^{k_0 k_1^x}$ is pseudorandom under the D-strong Diffie-Hellman assumption [42], and it can be evaluated obliviously using techniques for oblivious polynomial evaluation [26]. This OPRF is a trade-off between stronger assumption/smaller input domain and more efficient evaluation (see Table 4 for cost comparisons.).

2.2. Hashed Diffie-Hellman

The function $f_k^H(x) := H(x)^k$ with hash function H is a PRF under the idealized assumption that H produces uniformly random elements from a group $\langle g \rangle$ [43]. Implicitly when setting $g^a \leftarrow H(x)$, $f_k^H(x) := g^{ak}$ becomes a Diffie-Hellman value, hence we refer to this PRF as *HashDH*. f^H can be obliviously evaluated with a very simple "blinded exponentation" protocol, depicted in Figure 3, which is secure under the one-more gap Diffie-Hellman (OM-gapDH) assumption in the random oracle model (ROM).

	HashDH $H(x)^k$	2HashDH $H'(x, H(x)^k)$
Plain OPRF	[4], [5], [46]	[8]*, [10]
Partially oblv.	[19]	[12], [13]
Verifiable	–	[9] (+ NIZK) [13] (semi-H)
Correlated input	–	[13]
Adaptive key compr.	–	[14]
Proactive sec.	–	[12]
Updatable key	[19]	–
Distributed eval.	–	[12], [13]
Threshold eval.	–	[11]

TABLE 2: HashDH and 2HashDH OPRFs and which properties were already demonstrated for them. [8]* also hashes the first input of H'.

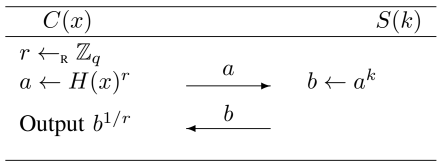


Figure 3: Blinded exponentiation for evaluating the HashDH PRF.

The idea behind this protocol, namely blinded DH exponentiation,[1] is not new and has a long history in cryptography: it is central to building blind signatures [44], [45], and it was used in the seminal work on password-based cryptography by Ford and Kaliski [46] and others [8], [43], [47], [48], [49]. We further explore analogies between OPRFs and blind signatures in Appendix B.2.

A series of works by Jarecki et al. [9], [10], [11], [14] shows that a slight variation of the above protocol called *2HashDH* [9], where the client adds an outer hash H' and outputs $f^{2H} := H'(x, H(x)^k)$, can be proven secure in the Universal Composability (UC) framework [40]. Security holds under the OM-gapDH assumption, and with both hash functions modeled as random oracles. 2HashDH is extensively used in the literature [10], [11], [12], [13], [14], [28] and has demonstrated to be versatile in terms of properties: the protocol allows for verifiable computation ($\mathcal{F}_{\mathsf{V-OPRF}}$ mentioned above) through efficient non-interactive zero knowledge proofs (NIZK), or a threshold version for sharing the OPRF key among multiple servers.

We refer the reader to Table 2 for further pointers on which oblivious evaluation properties have been already demonstrated for HashDH and 2HashDH. A variation of f^{2H} where the One-More RSA assumption is used instead of OM-gapDH is also included in [9], which we

1. Very recently, Jarecki et al. [29] investigated whether *multiplicative blinding* suffices for Hashed DH. This requires only fixed-base exponentiation and hence decreases the client's computational cost by a factor of $2 - 6$. However, the resulting protocol cannot satisfy standard OPRF security notions and is recommended to be used only when the correct value of the public key g^k is authenticated, and when the OPRF inputs are of high entropy. We show a close relation of their protocol to blind signatures in Section B.2.

summarize in Figure 15. The figure also demonstrates the similarities between 2HashRSA and Chaum's blind RSA signature scheme.

2.3. Dodis-Yampolskiy PRF

The Dodis-Yampolskiy (DY) PRF [50] is based on the Boneh-Boyen unpredictable function (originally introduced as a weak signature) [51]. The PRF requires a cyclic group $\mathbb{G} = \langle g \rangle$ of order q and is defined as

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

for a key $k \leftarrow_{\mathsf{R}} \mathbb{Z}_q$. Pseudorandomness is guaranteed by the q-DDH inversion (q-DDHI) problem [50] for polynomial-sized domains. The key aspect in obliviously evaluating f^{DY} is the use of an additively homomorphic encryption scheme such as Paillier [7], [16] or Camenisch-Shoup [22], [30].

We depict a blueprint of the evaluation protocol in Figure 4. The protocol is related to blind signing of secret keys [52]. A notable difference to oblivious evaluation of the NR PRF from homomorphic encryption is that, in the case of the DY PRF, the key pair belongs to the server. See, e.g., Camenisch et al. [22] on how to equip the protocol with efficient zero-knowledge proofs to protect against a malicious client. We note that the literature provides variants of the above blueprint, for example one where both the server and the client hold a key pair and computations are performed on ciphertexts encrypted with a combination of both public keys [7]. This construction then features full malicious security.

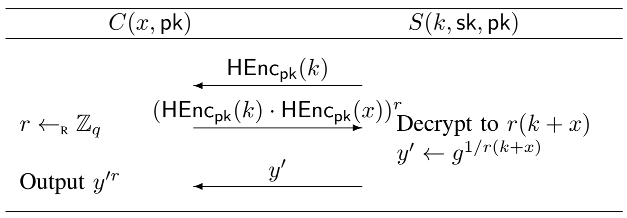

$C(x, \mathsf{pk})$
$S(k, \mathsf{sk}, \mathsf{pk})$
$\mathsf{HEnc}_{\mathsf{pk}}(k)$
$r \leftarrow_{\mathsf{R}} \mathbb{Z}_q$
$(\mathsf{HEnc}_{\mathsf{pk}}(k) \cdot \mathsf{HEnc}_{\mathsf{pk}}(x))^r$
Decrypt to $r(k+x)$
$y' \leftarrow g^{1/r(k+x)}$
y'
Output y'^r

Figure 4: Blueprint for oblivious evaluation of the DY PRF using homomorphic encryption scheme HEnc, omitting proofs of honest behavior.

The protocol in Fig. 4 requires a CRS comprising a safe RSA modulus with unknown factorization, and is secure under the Composite Decisional Residuosity (CDR) assumption if f^{DY} is pseudorandom in the corresponding domain [7].

A slight modification of the DY PRF, namely the function

$$f_k^{\mathsf{modDY}}(x, t) := H'(x, t, H(x)^{1/(k+t)}),$$

is considered in a recent work by Tyagi et al. [31]. Here, t is a public part of the message that is known to the server, implementing a useful property called *partial obliviousness* of OPRFs that we discuss further below. Making the exponent independent of secret x allows for efficient evaluation via blinded exponentiation, as used in 2HashDH explained above. Tyagi et al. demonstrate their OPRF secure under a new type of Diffie-Hellman inversion assumption.

2.4. Generic Techniques

Oblivious pseudorandom functions can be constructed from **oblivious transfer (OT)**, either by applying generic techniques for **secure multi-party computation (MPC)**, or directly from random OT (ROT). We explain both in more detail. Generic MPC schemes such as Yao's Garbled Circuits [53] can evaluate any function, in particular any PRF, obliviously w.r.t secret input x of the client and secret key k of the server. Put differently, any MPC protocol for securely evaluating a PRF on inputs x and k yields an OPRF according to Definition 1.

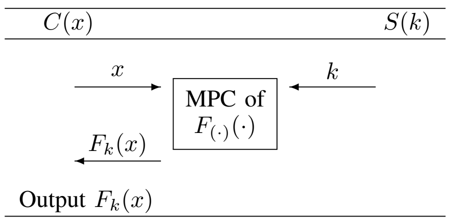


Figure 5: OPRF from secure MPC for any PRF F.

For example, the Advanced Encryption Standard's (AES) encryption function

$$f_k^{\mathsf{AES}}(x) := \mathsf{AES.Enc}(k, x)$$

is designed to produce uniform 128-bit strings on arbitrary 128-bit long inputs and hence is assumed to be a PRF [32]. There exist several improvements over generic MPC methods for securely evaluating the AES encryption circuit [32], [33], [54], directly yielding faster OPRF protocols for f^{AES}. Recently, block ciphers optimized for MPC emerged, such as LowMC [55], and the corresponding OPRF protocols [18] allow for efficient batching. We also mention here a line of work developing special-purpose OPRF protocols [6], [34], [35], [36], [56] optimized for implementing private set intersection (PSI). These OPRFs can be instantiated with any PRF, and leverage batched OT variants and special purpose hashing. We defer the reader to Appendix B.4 for further details.

Random OT (ROT) generically yields an OPRF. In $\binom{n}{1}$-ROT, the client obtains 1 out of n random values $k_1, \ldots, k_n$ generated by the server, without the client learning the other $n-1$ values, and without the server learning which value the client retrieved. Let x denote the index of the value that the client wants to receive. Then, the $\binom{n}{1}$-ROT corresponds to client and server obliviously evaluating PRF $F_k(x) := k_x$ with $k := (k_1, \ldots, k_n)$, where k_x denotes the x-th element of k [6].

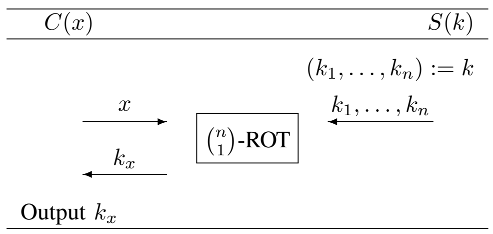


Figure 6: OPRF with input domain $\{1, \ldots, n\}$ from random OT.

Consequently, a $\binom{n}{1}$-ROT protocol with arbitrarily large n, also often called an OT extension protocol [39], yields an OPRF with unlimited input domain $\{0,1\}^*$ [6]. In Appendix B we discuss the relation between OPRFs, blind signatures and OT in more depth.

Another generic way to build an OPRF is to use a **unique blind signature** scheme [44], [45] $\mathsf{BSIG} = (\mathsf{Gen}, \mathsf{Blind}, \mathsf{Sign}, \mathsf{Unblind}, \mathsf{Vfy})$. This is a signature scheme with a deterministic signing algorithm (uniqueness of signatures), and where the signer does not learn which message he signed (blindness). If BSIG produces pseudorandom signatures, we can interpret the secret key of key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\mathrm{R}} \mathsf{Gen}$ as PRF key $k \leftarrow \mathsf{sk}$ and directly obtain an OPRF for PRF $\mathsf{Sign}_k()$, as depicted below. The two known unique blind signature schemes in the literature [44], [45] yield particular HashDH OPRFs already described in Section 2.2 (see Appendix B.2 for details).

$$\begin{array}{lcr}
C(x) & & S(k) \\
\hline
(\bar{x}, r) \leftarrow_{\mathrm{R}} \mathsf{Blind}(x) & \xrightarrow{\bar{x}} & \\
\sigma \leftarrow \mathsf{Unblind}(\bar{\sigma}, r) & \xleftarrow{\bar{\sigma}} & \bar{\sigma} \leftarrow \mathsf{Sign}_k(\bar{x}) \\
\text{Output } \sigma & &
\end{array}$$

Figure 7: OPRF from a unique blind signature scheme $(\mathsf{Gen}, \mathsf{Blind}, \mathsf{Sign}, \mathsf{Unblind}, \mathsf{Vfy})$.

3. Properties

We now turn to describing additional properties of OPRFs. For this whole section, we recommend frequently checking out Table 3, which gives further pointers to OPRF protocols conforming with the notions presented herein.

3.1. Partially-oblivious PRFs

While OPRFs were originally introduced as a fully oblivious primitive, subsequent constructions required revealing part of the client's input x to the server [19]. These constructions are called partially-oblivious PRF (initially, the term *tweak visibility* was also used) and are purely application-motivated: they give the server sufficient control to establish a rate limit for the number of evaluation requests made by the client, with respect to a specific public part of the input. We denote the private part of the input as x_{priv}, and the public one as x_{pub}. The goal of the joint protocol is to allow the client to obtain a value $f_k(x_{\mathsf{priv}}, x_{\mathsf{pub}})$, but now with the server learning x_{pub}:

Definition 2 (Partially-oblivious pseudorandom function)**.** A two-party protocol π between a client and a server is a *partially-oblivious pseudorandom function (pOPRF)* if there exists some PRF family f_k, such that π privately realizes the following functionality:

- Client has inputs $x_{\mathsf{priv}}, x_{\mathsf{pub}}$; Server has key k and public input x_{pub}.
- Client outputs $f_k(x_{\mathsf{priv}}, x_{\mathsf{pub}})$; Server outputs nothing.

Formal versions of this property have been defined in UC [12], [13] and with game-based models [19], [31].

Constructions. A common construction for partially-oblivious PRFs uses the (2)HashDH in a bilinear map setting, leveraging the pairing to combine both input values. The original construction proposed as part of the Pythia protocol [19] uses a pairing e to compute $f_k(x_{\mathsf{priv}}, x_{\mathsf{pub}}) := e(H_1(x_{\mathsf{pub}}), H_2(x_{\mathsf{priv}}))^k$. While x_{pub} is known to the server, the pairings second input $H_2(x_{\mathsf{priv}})$ is send blindly as $a \leftarrow H_2(x_{\mathsf{priv}})^r$ to the server. The server returns $b \leftarrow e(H_1(x_{\mathsf{pub}}), a^k)$ to the client, which unblinds the value into $y \leftarrow b^{1/r}$. For the 2HashDH variant – which is needed to get UC security – the unblinded value is simply hashed again with both inputs.

Recently, Tyagi et al. [31] considered the partially-oblivious PRF

$$f_k^{\mathsf{modDY}}(x_{\mathsf{priv}}, x_{\mathsf{pub}}) := H'(x_{\mathsf{priv}}, x_{\mathsf{pub}}, H(x_{\mathsf{priv}})^{1/(k+x_{\mathsf{pub}})}),$$

which can be evaluated with only three exponentiations, hence omitting the need for pairings. The idea of this construction is that the server augments the key k of 2HashDH with the public input. Since this must occur in a non-malleable way, the exponent is inverted and the resulting PRF resembles the Dodis-Yampolskyi PRF.

Related to this idea, there is a simple trick to derive an OPRF output from a partially known input, which could generically be combined with all (fully-blind) OPRF constructions, yet so far it has only been instantiated in combination with (2)HashDH [13], [57], [58]. Therein the public input is used to derive an x_{pub}-specific key using a standard pseudorandom function PRF. That is, a partially-oblivious PRF $f_k(x_{\mathsf{priv}}, x_{\mathsf{pub}})$ can be derived from a fully-oblivious PRF f' and a standard PRF as

$$f_k(x_{\mathsf{priv}}, x_{\mathsf{pub}}) := f'_{\mathsf{PRF}(k, x_{\mathsf{pub}})}(x_{\mathsf{priv}}).$$

Thus, technically, the server uses a different key for every public input. The advantage of this approach is that it allows to add partial-blindness to (2)HashDH without requiring bilinear maps.

3.2. Verifiability & Committed Inputs/Outputs

We now review different ways of establishing guarantees on the inputs and outputs of an OPRF protocol.

Verifiable OPRF. A *verifiable* OPRF (V-OPRF) refers to the concept of correct output verifiability. That is, it allows the client to be convinced that she has received the PRF value $f_k(x)$ with k denoting the server's key. Definition 1 is already verifiable, as it *guarantees* that the output of the client is $f_k(x)$ whenever the client inputs x and the server inputs k. Verifiability can be achieved by handing the client a "public key" version of the server's key k, which can be used not only to verify correctness of PRF evaluations but also equality of the server's key among different runs of the protocol.

This property can easily be realized in the DL-based setting, by letting the client obtain a commitment g^k to the PRF key k, and requiring the server to provide zero-knowledge proofs of correct evaluation w.r.t. k, or using some construction-specific verification method, e.g., when using blind-signature-based constructions or relying on pairings. Verifiable constructions for NR, DY and (2)HashDH-based OPRFs have been proposed so far; see Table 3 for an overview. We note that some works [4], [13] present verifiable definitions of OPRFs but only achieve them by restricting to semi-honest servers. As semi-honest

	PRF	Batching	Multi-point	Partially oblivious	Verifiable	Committed	Proactive sec.	Updatable	Distributed	Threshold	Special (Sect. 3.8)
FIPR05, 1st [4]	NR	○	●	○	○	○	○	○	○	○	○
HL08 [5]	NR	○	●	○	●	○	○	○	○	○	○
JKK14 [9]	NR	○	●	○	●	○	○	○	○	○	○
Hazay15 [26]	NR variant	○	●†	○	●	○	○	○	○	○	○
JL09 [7]	DY	○	●	○	●	●^{i}	○	○	○	○	○
CL17 [22]	DY	○	●†	○	○	●^{o}	○	○	○	○	○
MPRSY20 [30]	DY	●	●	○	●	○	○	○	●	○	○
TCRSTW21 [31]	DY variant	○	●	●	●	○	○	○	○	○	○
JL10 [8]	2HashDH	●†	●	○	○	○	○	○	○	○	○
JKK14 [9]	2HashDH/RSA	○	●	○	●	○	○	○	○	○	○
ECSJR15 [19]	HashDH	○	●	●	●	○	○	●	◍	●	○
JKKX16 [10]	2HashDH	○	●	○	○	○	○	○	○	○	○
JKKX17 [11]	2HashDH	○	●	○	○	○	○	○	◍	●	○
DGSTV18 [28]	2HashDH	●†	●†	○	●	○	○	○	○	○	○
Lehmann19 [57]	HashDH	●♣	●	◍	○	○	○	◍	○	○	●
BFHLY19 [12]	2HashDH	○	●	●	○	○	●	○	●	○	○
DHL22 [13]	2HashDH	○	●	●	○	○	○	○	●	○	●
KKRT16 [6]	Any	●✶	○	○	○	○	○	○	○	○	○
KMPRT17 [34]	Any	○	○	○	○	○	○	○	○	○	●
JKR18 [58]	Any	○	●◇	●	●	○	○	●	◍	●	○

TABLE 3: Properties of OPRFs from the literature.

● = applicable	✶ = with related keys	♣ = for same input
◍ = trivially implied	† = enforcing same key	sh = semi-honest server
○ = not satisfied	◇ = input-dependent key	i = input, o = output

behavior of the server implies verifiability by assumption, their constructions should not be considered as verifiable.

Committed In- & Outputs. Since OPRFs are often used as building blocks in more complex protocols, it might be necessary to ensure that they are run on certain well-formed values, or to allow subsequent computations for the blindly derived outputs. Both can be realized via cryptographic commitments, such as Pedersen commitments.

For committed inputs, this slightly weakens the secrecy property of OPRFs from full blindness (i.e., where the server learns nothing about the input) to learning a commitment $c \leftarrow com(x; r)$ of the client's input x and with r denoting the randomness for the commitment. This commitments can serve as "glue" to other parts of the protocol, e.g., the client can prove that she invoked the OPRF on a value that was the result from a previous step in the protocol.

Definition 3 (OPRF with Committed Inputs)**.** A two-party protocol π between a client and a server is an OPRF with committed inputs if there exists some PRF family f_k such that π privately realizes the following functionality:

- Client has input x, r; Server has key k and input $c \leftarrow com(x; r)$.
- Client outputs $f_k(x)$; Server outputs nothing.

OPRFs with committed outputs instead have full blindness for the inputs, but the server learns a commitment $c \leftarrow com(y; r)$ of the client's output $y := f_k(x)$. A high-level protocol can now use this commitment to ensure that the client correctly uses the blindly computed OPRF value y in its subsequent computations.

Definition 4 (OPRF with Committed Outputs)**.** A two-party protocol π between a client and a server is an OPRF with committed outputs if there exists some PRF family f_k such that π privately realizes the following functionality:

- Client has input x; Server has key k.
- Client outputs $(y \leftarrow f_k(x), r)$; Server outputs $c \leftarrow com(y; r)$.

In terms of constructions, both commitment approaches require that the blinded in- and outputs of the underlying PRF preserve a certain (algebraic) structure, as is the case for the NR and DY PRFs. While we find a DY-based OPRF with committed input [7] and committed output [22] in the literature, these properties have not yet been investigated for NR-based PRFs. The (2)HashDH-based constructions are not amendable to this setting, as the hash "destroys" the structure of in- and outputs, and the same holds for oblivious evaluations of non-algebraic PRFs such as, e.g., f^{AES}.

3.3. Updatable OPRFs

Key rotation, i.e., periodically changing the secret key, is a common security measure in order to minimize the risk and impact of key exposure. An OPRF is called *updatable* if there is an efficient (possibly interactive) procedure that lets the client update previously computed PRF values to a new key. We note that such updatability is independent of the evaluation process and hence rather a property of the PRF. To realize such an update procedure, one requires controlled access to the underlying key, and thus again a certain algebraic structure from the PRF value. The natural PRF construction that satisfies these requirements is $f_k(x) := H(x)^k$, and in the context of OPRFs the feature of updatability was first proposed (for the pairing-version of that PRF) in [19].

The main idea is as follows: If the server wants to rotate from the current key k to a new k', it issues the compact token $\Delta = k'/k$ to the client. The client can then efficiently update his stored PRF value $f_k(x) := H(x)^k$ by raising it to Δ.

This approach is compatible with all OPRFs that produce outputs of the form X^k (for arbitrary X), but unfortunately does not work with the more popular 2HashDH variants that offer stronger UC-security. In fact, it is an interesting open problem how to construct an updatable *and* UC-secure OPRF.

Another interesting update feature is *proactive security* (a term also referred to as *recovery from compromise* or *security against transient corruptions*[2]) if it guarantees a recovery strategy that allows a server to refresh its keys so that it can securely re-initialize after a compromise. This works through key rotation [19], where the server switches to a fresh key and provides a short update token that allows to update all previous PRF outputs. If the server is distributed (see below), techniques from proactively secure secret sharing apply [11], [12]. Proactive security can be seen as the "opposite" of forward secrecy, where compromise of the OPRF key can only impact security of evaluations *until* the next recovery procedure is executed.

3.4. Distributed & Threshold ORPFs

While the original OPRF definition [4] only involves one client and one server, several extensions exist that distribute the server role to several entities. The main goal of such distribution is to increase resilience against key compromise. This is particularly important when the OPRF is used to protect low-entropy inputs, such as passwords, where a full key exposure would allow to recover the inputs of acquired PRF values through brute-force attacks.

The DH-based constructions naturally lend themselves to both distributed and threshold versions, by simply applying either additive sharing of the key or Shamir's secret sharing respectively. The latter realizes a (t, n)-*threshold* OPRF [11], [19], by splitting the secret key k over a set of n servers, such as any subset of t servers can jointly compute the PRF value $f_k(x)$. Consequently, computation of the PRF value now becomes an interactive protocol of the client and at least t servers. As long as no more than t servers are corrupted (at the same time), the adversary learns nothing about the key and thus cannot brute-force learned PRF outputs. If $t = n$ this is called a distributed PRF [12]. Jarecki et al. [14] show how to use a *share conversion* technique due to Cramer et al. [59] to thresholdize a partially oblivious PRF. Their construction however is not fully generic as it requires the underlying OPRF to derive keys from a PRF, and it is only efficient in small settings since it requires $\binom{n-1}{t}$ exponentiations on the client side.

2. *Permanent vs transient:* a permanent corruption is for the party's lifetime (i.e., once a server is corrupted it will always be controlled by the attacker), whereas that in a transient corruption the attacker controls the server only for a specific amount of time.

So far, distributed and threshold schemes mainly build upon (2)HashDH and are nicely compatible with other properties that were realized for that line of constructions, such as partial-blindness and verifiability. While generic MPC could of course be used to distribute all OPRF constructions, realizing dedicated efficient versions of NR-based scheme is still an open problem.

3.5. Choice of PRF Key

In many OPRF applications, such as PSI or password-based cryptography, one needs to obliviously evaluate a PRF more than once. If the OPRF allows for consecutive evaluations w.r.t the same key, then we call the OPRF *multi-point*, otherwise we call it *single-point*.

Definition 1 models a *multi-point* OPRF (also referred to as *multi-query* [34], *multi-evaluation*, or *multi-session* [22]), by allowing the server to provide an arbitrary OPRF key for each evaluation. Most OPRFs in the literature are multi-point (see Table 3), and some even enforce usage of the same key in consecutive evaluations [22], [26]. We can further distinguish between *adaptive* queries, where the client can decide on the value of each query after receiving the answer to previous queries (as modeled by Def. 1), and a *non-adaptive* (also called *static*) setting, where the client must provide all queries before receiving any evaluation.

Single-point OPRFs (also referred to as *one-time* [60] or *single-evaluation*) do not take an OPRF key as input. Instead, a uniformly (mostly, with only few exceptions [6]) key is given to the server as output.

Definition 5 (Single-point oblivious pseudorandom function)**.** A two-party protocol π between a client and a server is a *single-point OPRF* if there exists some PRF family f_k, such that π privately realizes the following functionality:

- Client has input x;
- Client outputs $f_k(x)$; Server outputs k.

Obviously, a single-point OPRF cannot be evaluated twice with respect to the same PRF key k (except with negligible probability). This limits the applicability of such OPRFs, and indeed, all single-point OPRFs from the literature [6], [34], [35], [54], [61], [62], [63] are used exclusively in the context of Private Set Intersection (PSI). An interesting side fact, which we develop further in Appendix B, is that the relationship between single-point and multi-point OPRFs is analogous to the one between Random OT and OT [6], [60]. It is thus not surprising that the seminal Random OT extension protocol of Ishai et al. [39] is used as single-point batched OPRF by Kolesnikov et al. [6].

All algebraic OPRFs (i.e., based on the NR PRF, the DY PRF, or Hashed DH) are multi-point, and hence the PRF key is chosen by the server. Additionally, some OPRFs enforce usage of the *same* key in consecutive (or batched) runs of the protocol, which works for the NR-PRF [26], the DY PRF [22], and also 2HashDH [28]. To really enforce same key usage (and not only deploy some sort of "soft" enforcement, where the client discards all outputs that do not verify with respect to a public key), an OPRF with same-key enforcement can be easily built for the DY PRF as in Figure 4: the client always uses the same $\mathsf{Enc}_{\mathsf{pk}}(k)$ in each evaluation, and proves correct behavior in zero-knowledge [22].

Lastly, there exist transformations from single-point to multi-point OPRFs based on Cuckoo hashing [34].

3.6. Batched OPRFs

Batching improves the efficiency of *multiple parallel* OPRF executions either with the same [8] key yet different inputs, or different keys yet same inputs [57], or correlated keys [6]. From a batched OPRF protocol that lets a client compute ℓ PRF values we expect a lower complexity than that of performing ℓ successive runs of the corresponding single evaluation OPRF protocol.

The main setting for batched OPRFs is for batched inputs, i.e., the client wishes to obtain several PRF values for different input values at once. This is a particularly desirable feature in large-scale private set intersection protocols, e.g., for privacy-friendly contact discovery [18]. So far, all constructions for batched inputs are based on generic OPRF-constructions from symmetric primitives [6], [35], [54], [62], [63].

In contrast, for realizing batching for different keys, i.e., the client wants to obtain ℓ PRF values for ℓ different keys for the same input x, (2)HashDH is a natural candidate as it allows to simply reuse the blinding input value in several responses [57]. See Figure 8 below which compares the standard and batched constructions.

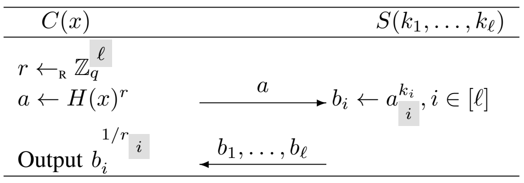


Figure 8: Batched and unbatched HashDH. In an unbatched run, r and a are ℓ-dimensional vectors and hence the client needs to send ℓ group elements instead of one as in the batched version.

3.7. Weak OPRFs

Ideally, an OPRF protocol leaks no information beyond the client learning the evaluation. But such a rigid definition is often too strong, and it does not allow to prove security of many efficient OPRF protocols, for example ones where the client applies a hash to derive the final PRF output. The notion of *weak OPRF*[3] (also called *relaxed* OPRF) was also coined by Freedman et al. [4]. It allows the client to learn additional key-dependent information. This additional information may not compromise pseudorandomness of yet unqueried inputs [4, Def. 6]. For example, consider an OPRF that reveals the first bit of the PRF key, or one that reveals the preimage of a final key derivation function. Such OPRFs are still "secure enough" for most applications. Prominent examples are 2HashDH OPRFs (see Table 2 for references) and a special OPRF of Kolesnikov et al. [6], which serves as the basis of many OPRFs tailored for private set intersection (see Appendix B.4).

3. Not to be confused with weak PRFs. A *weak PRF* is pseudorandom on randomly chosen inputs. For example, $f_k(x) := x^k$ is a weak PRF, but not a PRF: given $f_k(x)$ one can easily compute, e.g., $f_k(x^2)$. Transformations from weak PRFs to PRF exist in the ROM [43]. We are not aware of any formal definitions of oblivious evaluation protocols of weak PRFs, but note that [15] implicitly evaluates weak PRF $H(x^k)$ in an oblivious way. See Section 6 for further discussion.

3.8. Specialized properties

We now explain several "niche" properties that were introduced to serve particular applications, but have not (yet) found use beyond.

A *programmable* OPRF (OPPRF) [34] allows the server to "program" the output of the PRF f on a limited number of inputs. More formally, in an OPPRF the server holds a constant-sized set of point-value pairs (x_i, y_i). Then, the server generates an OPRF key k such that $\mathsf{PRF}_k(x_i) = y_i$ and PRF_k is pseudorandom everywhere else. The client learns the PRF output as before, but, importantly, is not able to distinguish whether his input was one on which the PRF was programmed by the server. OPPRFs allow for efficiency gains in modern PSI protocols [34], [35], [62], [63] (see Appendix B.4). OPPRFs can be constructed generically from OPRFs [34], [62].

The OPRF in [57] provides two new features: First, it expresses and realizes OPRFs for a *3-party setting*, where the party blindly invoking the OPRF and the one receiving the unblinded output are different entities. For this, the OPRF is moved to a public-key setting where the blinding and evaluation are done w.r.t. a public blinding key *bpk* of the receiving entity. Only the receiver knows the corresponding secret key *bsk* and can unblind the derived PRF value. This feature is useful when the correlation between in- and outputs must not be known to a single party; e.g., when the OPRF is used to create pseudonymized data in a way that no entity can link the original identifier with its derived pseudonym.

Another unique feature introduced in [57] is *convertability*. This allows a blind conversion from an already received PRF value $y = f_k(x)$ into $y' = f_{k'}(x)$. This blind conversion can only be performed through the server that knows both k and k'. The privacy properties are maintained through conversion, i.e., the server neither learns y, y' or the underlying x for which the PRF values are derived. This was again proposed in the context of pseudonymization, but the feature also naturally lends itself to key rotation, as a server can blindly rotate all previously computed PRF values under key k towards a fresh key k'.

The concept of *extendable* OPRFs was introduced in [13] and allows that certain blinded inputs can be re-used in the second evaluation. That is, after receiving a PRF value $y = f_k(x_{\mathsf{pub}}, x_{\mathsf{priv}})$ the client and server can engage in a subsequent run of the protocol where the client obtains $y = f_k(x'_{\mathsf{pub}}, x'_{\mathsf{priv}}, x_{\mathsf{priv}})$. The second value is evaluated on fresh (partially-blind) inputs $x'_{\mathsf{pub}}, x'_{\mathsf{priv}}$ but also re-uses x_{priv} from the previous run. This feature can be useful in password-based protocols, where the first PRF value serves for explicit password-verification whereas the second PRF value derives a password-depended key from the verified password.

3.9. Discussion

On choosing between the different OPRF constructions. As explained in Section 2, there are two groups of OPRFs in the literature: algebraic constructions (evaluating the NR-, DY- or HashDH PRF) and OT-based constructions (used mostly to evaluate block-cipher based PRFs such as AES encryption). OT-based OPRFs are a good choice if many evaluations need to be computed, in particular if the application allows for randomly chosen keys (single-point OPRFs), and if bitstrings as PRF values are sufficient. Table 5 can be consulted to pick an appropriate OT-based OPRF.

Algebraic OPRFs are computationally more expensive than OT-based OPRFs, but produce pseudorandom group elements as output and offer a variety of interesting properties as explained in this section. Table 3 helps in picking an OPRF with the desired property combination, and Table 4 depicts computational and commmunication costs of algebraic OPRFs. Regarding the type of PRF, while NR-based OPRFs are conceptually simple, they do not allow for much flexibility and have thus found only very little adoption in recent years. HashDH-based constructions are the most abundant in the literature and offer the broadest variation of properties – if one wants to construct an OPRF with new properties, HashDH-based OPRFs are a good starting point! HashDH-based OPRFs also have optimal communication complexity (1 group element in each direction) per single evaluation. Still, for some properties such as committed input/output, DY-based OPRFs are currently the only available option, however at a significant efficiency loss due to use of higher order groups with unknown factorization. In settings where quantum security matters, although none of the algebraic OPRFs are quantum secure, HashDH-based OPRFs information-theoretically blind the input of the user, and hence protect it even against quantum computers. This is not the case for NR- and DY-based constructions, where a discrete logarithm solver can recover the user's input. Lastly, we note that some algebraic OPRFs should be handled with care, e.g., ones that require more interaction [7], [9], [26] or do not provide security against malicious servers [4], [13], [22]. Table 4 marks such cases in red.

Regarding benchmarking of resource/bandwith settings in which algebraic and OT-based OPRFs outperform each other, we refer the reader to the works of [18], [33], [54], [64].

On potentially conflicting properties. It can be seen from Table 3 that none of the non-hash-based OPRFs, namely the NY/DY-based ones, provide updatability. The crucial difference between NR/DY-based OPRFs and HashDH-based ones is that, in HashDH, the key-dependent exponent is simply k, hence it is *independent* of the user's input and thus easy for the server to generate a universal update token. For the DY PRF, the exponent is $1/(k+x)$ and hence dependent on a value that the server does not know, and similarly for the NR PRF. Manipulating the key in PRF values in such a way that the user cannot derive related evaluations seems inherently hard – at least as long as we require the manipulation procedure to be non-interactive.

Another conflict arises when one wants to build a universally composable OPRF (see upcoming Section 4 about security) that supports key rotation, providing the user with update tokens that let her update old PRF values to the new key. As explained above, the only choice for key rotation seems a hash-based OPRF, such as HashDH. Yet proving universal composability of HashDH seems to require an "outer hash", as otherwise very inefficient techniques such as proof of correct hash computation need to be added on top. However, the outer hash inherently limits the client's "access" to the key, as it makes it impossible to perform any arithmetic operations on the hashed value.

Lastly, the generic way to construct partially-oblivious PRFs using $\mathsf{PRF}(k, x_{\mathsf{pub}})$ as the key, as described in Section 3.1, somewhat conflicts with the goal of verifiability. This is because in that case, the OPRF is computed with many different keys, requiring the server to additionally provide x_{pub}-specific "public keys", and also proving that these keys have been correctly derived from k. The latter might require proving correct evaluations of an AES circuit, depending on the choice of PRF.

4. Formalizing OPRF security

We explain how to formalize security of an OPRF using the two options available, namely simulation-based and game-based. We also describe special security properties, such as proactive security, and how to capture them as well as the properties of the previous section in the different models. Practically-oriented readers might want to skip this section, and directly jump to applications (Section 5).

4.1. Simulation-based security

Simulation-based security follows a real world – ideal world paradigm. In a nutshell, an OPRF is considered secure if it "behaves" as if it was carried out by a trusted party, and the protocol transcript can be simulated without knowledge of secret inputs. A good introduction to simulation-based security is offered by Lindell [65].

Defining simulation-based security of OPRFs requires formulating how an "ideal" OPRF looks like: how would a fully-trusted party $\mathcal{F}$, talking to a client and a server, execute the task? Definition 1 already shows the two main aspects that $\mathcal{F}$ needs to implement:

- $\mathcal{F}$ implements a truly random function;
- $\mathcal{F}$ only lets the client obtain an evaluation if the server agrees to participate.

As OPRFs are often used as building blocks in cryptographic protocols, strong composability and concurrency guarantees are desirable. Unfortunately, Definition 1 does not ensure secure concurrent executions of the OPRF [9]. A simulation-based framework which offers particularly strong composability and concurrency guarantees is the Universal Composability framework (UC, [40]), and it is hence not surprising that it is frequently used as a method of formalization in the OPRF literature [9], [10], [11], [12], [13], [14], [21], [22], [28], [29]. All these works provide differing OPRF functionalities, with special properties that make it hard to understand the essence of

The functionality assigns [1] random values $F_S(x) \leftarrow \{0,1\}^\lambda$ for yet undefined $F_S(x)$.

On $(\textsc{Init})$ from [2] server S, send $(\textsc{Init}, S)$ to adversary $\mathcal{A}$. Ignore all subsequent $\textsc{Init}$ queries.

On [3] $(\textsc{Eval}, \mathsf{ssid}, x)$ from any client C, do:

- send [4] $(\textsc{Eval}, \mathsf{ssid}, S)$ to S and $\mathcal{A}$;
- record (C, ssid, S, x).

On $(\textsc{Proceed}, \mathsf{ssid})$ from [5] server S, do:

- send $(\textsc{Proceed}, \mathsf{ssid})$ to $\mathcal{A}$ and [9] receive back (ssid, S');
- retrieve record (C, ssid, S, x);
- [7] abort if $S \neq S'$ and S is honest;
- output [6] $(\mathsf{ssid}, F_{S'}(x))$ to C.

On [8] $(\textsc{OfflineEval}, x)$ from $\mathcal{A}$, if S is corrupt then send $F_S(x)$ to $\mathcal{A}$.

Figure 9: UC functionality $\mathcal{F}_{\mathsf{OPRF}}$, without any special properties.

the functionality. Therefore, in Figure 9, we give the core code that they all comprise.

Outputs of $\mathcal{F}_{\mathsf{OPRF}}$ are consistent and indistinguishable from random through maintaining a function table $F_{(\cdot)}(\cdot)$, assigning random values wherever no value is assigned yet ([1]). Hence, any OPRF indistinguishable from $\mathcal{F}_{\mathsf{OPRF}}$ implements a pseudorandom function. $\mathcal{F}_{\mathsf{OPRF}}$ is "keyed" by the server identity S ([2]) and, in unattacked sessions, computes function $F_S(\cdot)$. When a client wants to evaluate the PRF ([3]) on input x, privacy of x is ensured by informing the adversary $\mathcal{A}$ about the evaluation but not leaking x ([4]). The client only receives output if the server agrees in the evaluation ([5], an evaluation is identified by a unique ssid). The output received by the client is either $F_S(x)$ in an honest run of the protocol, or $F_{S'}(x)$ in case the adversary (= the corrupt server) decides to use a different PRF key S' ([7]) that is not equal to S. Finally, in case the server is corrupt, the adversary can freely evaluate the PRF via interface $\textsc{OfflineEval}$ ([8]). Finally, the adversary can always mount a DoS attack by not replying anymore ([9]). It should be clear from this description that $\mathcal{F}_{\mathsf{OPRF}}$ specifies not only input-output behavior of an OPRF but also all attacks that the OPRF admits. The motivation behind allowing attacks in the first place is efficiency in the realizing OPRF protocol.

One drawback is that definitions following Figure 9 cannot be realized by AES-based OPRFs [6], [18], [35], [36], [54], [56], [61], [62], [63], since pseudorandomness of AES is based on heuristics.[4] For the other three main PRF classes from the previous section, UC-secure OPRFs exist (cf. Table 4). Several variants of the PRF $H(x)^k$, such as $H'(x, H(x)^k)$, owe the outer hash H' to UC security, since this hash enables efficient extraction of inputs. Since the hash is essentially required for UC security, all papers that do algebraic manipulations of the key, such as key rotations, instead use game-based security notions [8], [19], [57], which we detail below.

We now give explanations on how to extend Figure 9 to capture properties from the previous section.

4. Camenisch et al. [22] give a UC OPRF functionality which differs from Figure 9: their $\mathcal{F}_{\mathsf{OPRF}}$ has a hard-coded PRF which is used to generate outputs – similar to Definition 1. Indistinguishability from their $\mathcal{F}_{\mathsf{OPRF}}$ hence only implies unpredictability. On the plus side, a definition with hard-coded PRF can be used to argue simulation-based security even of AES-based OPRFs, under the assumption that AES is a PRF.

- **Partial obliviousness:** modify ([4]) to let $\mathcal{F}_{\mathsf{OPRF}}$ leak the public part of the input to the server [12].
- **Verifiability:** remove instruction [7] and remove S' from [9]. Namely, do not let $\mathcal{A}$ switch to $S' \neq S$ even in case the server gets corrupted after an honest $\textsc{Init}$ phase [9].
- **Committed inputs/outputs:** $\mathcal{F}_{\mathsf{OPRF}}$ gets an additional $\textsc{Commit}$ interface that allows the client to commit to his inputs or his outputs, where $\mathcal{F}_{\mathsf{OPRF}}$ ensures consistency with actual inputs and outputs [22].
- **Key rotation:** Due to randomly chosen outputs, there is no natural way to capture key updates or objects such as update tokens within $\mathcal{F}_{\mathsf{OPRF}}$. And indeed, the literature does not provide any definition or construction of a UC-secure OPRF with key rotation.
- **Proactive security:** $\mathcal{F}_{\mathsf{OPRF}}$ gets an additional "uncorrupt S" interface, which lets it treat server S as honest again in instructions [7] and [8] [12].
- **Distributed/threshold:** the $\textsc{Init}$ and $\textsc{Proceed}$ interfaces have to be called by all servers, or a threshold of them. $\mathcal{F}_{\mathsf{OPRF}}$ decides about honest or corrupt evaluation based on how many servers are corrupt [11], [12], [13].
- **Programmability:** Modify the $\textsc{Init}$ interface to let the server provide a list of point-value pairs to program into $F_S(\cdot)$ [62].
- **Multi-point:** $\mathcal{F}_{\mathsf{OPRF}}$ already implements a multi-point PRF, since repeated calls to $\textsc{Eval}$ allow evaluation of the same function $F_S(\cdot)$.
- **Single-point:** this can be captured by dropping the $\textsc{Init}$ query and letting $\mathcal{F}_{\mathsf{OPRF}}$ choose a fresh key S for each $\textsc{Eval}$. Additionally, $\mathcal{F}_{\mathsf{OPRF}}$ must offer the involved server an $\textsc{OfflineEval}$ interface with respect to that key. Single-point OPRF definitions do not exist in the literature, as they only implicitly appear in works based on the PSI protocol of Kolesnikov et al. [6].
- **Batching:** Since $\mathcal{F}_{\mathsf{OPRF}}$ does not demand anything about the efficiency of the realizing protocol, batching cannot be captured in a UC definition.
- **Weak OPRF:** $\mathcal{F}_{\mathsf{OPRF}}$ can be realized by weak OPRFs, since it already enforces pseudorandomness of *other* PRF values via instruction [1]. Additional leakage about the key is allowed as long as a simulator can fabricate indistinguishable information.

Finally, we mention a commonly used proof technique that allows to tweak efficiency of UC-secure OPRFs. Namely, Jarecki et al. [9] added a so-called *ticketing mechanism* to OPRF functionalities. This is a technique used in simulation-based proofs which dispenses with the need to "on-line" extract the user's input during the execution (note that the existence of a simulator already implies an efficient extractor of the input message). Ticketing avoids heavy cryptographic mechanisms and costs for, e.g., extractable proofs of knowledge when building a simulator for proving the security of a given OPRF protocol. Essentially, each evaluation results in generation of a ticket, and each PRF value given out reduces the number of tickets by one, making sure that the overall number of values given out does not exceed the number of protocol runs. A similar method to the ticketing mechanism for

OPRFs has already been considered in the context of blind signatures in the UC model [66].

4.2. Game-based security

Another way to define security of OPRFs is to inspect various aspects of the OPRF via experiments formulated as games between an adversary and a challenger. For example, security against a malicious client can be checked by challenging an adversary to predict an unqueried PRF value (one-more unpredictability, [19], [57]), or to distinguish it from random (pseudorandomness, [57]). More detailed, to break one-more unpredictability of an OPRF protocol, the adversary $\mathcal{A}$ plays an interactive game with a challenger who randomly samples a PRF key $k \leftarrow_R \mathcal{K}$ from key space $\mathcal{K}$. $\mathcal{A}$ gets access to a "transcript oracle" PRFrsp and can use it to receive polynomially-many PRF values together with the evaluation transcript. $\mathcal{A}$ then needs to output one more input-output pair of the PRF. We exemplarily depict this in Figure 10, but note that it only works for 1-round OPRF protocols Π. For more complex protocols, the transcript oracle PRFrsp would need to be more involved. The choice of notion (unpredictability is weaker than pseudorandomness) depends on the application: if the PRF value is used as a credential, unpredictability might be sufficient [19]; if the PRF value is used as a pseudonym [57], pseudorandomness seems required.

```
Game $\mathsf{Exp}^{\texttt{one-more unpred}}_{\mathcal{A},\Pi}$
$k \leftarrow_R \mathcal{K},\ c \leftarrow 0$
$(x_1, y_1), \ldots, (x_n, y_n) \leftarrow \mathcal{A}^{\mathsf{PRFrsp}_k(\cdot)}$
If $\exists i, j \in [n], i \neq j$ with $(x_i, y_i) = (x_j, y_j)$ return $0$
Return $(\wedge_{i=1,\ldots,n}\, y_i = \mathsf{PRF}_k(x_i) \wedge c < n)$

Oracle $\mathsf{PRFrsp}_k(\cdot)$:
on input $x$ set $c{+}{+}$ and return $\mathsf{PRFrsp}_k(x)$
```

Figure 10: One-more unpredictability game for a 1-round OPRF protocol Π evaluating PRF PRF, where the server's response is computed with function PRFrsp. If $\mathcal{A}$'s advantage in winning this game is negligible, then Π is one-more unpredictable.

To check security of an OPRF protocol w.r.t. malicious servers, we can define a corresponding experiment that checks *obliviousness* of the protocol, resembling semantic security of an encryption scheme: the adversary provides two inputs, one of which is randomly selected by a challenger and used to generate the protocol transcript. The adversary wins the game if it can tell which input was selected. See Lehmann [57] for a formal definition.

Only few OPRFs in the literature are proven secure in a game-based model [8], [19], [31], [57], and the only properties formalized so far are proactive security, verifiability and key rotation [19].

5. Applications

A common design paradigm to leverage an OPRF is to let a client[5] compute a high-entropy cryptographic object (e.g., a key, or a token) from a low-entropy input (e.g., a password, a username, an identifier, or a file). The fact that the computation is assisted by one or many servers enables protocols that are lightweight on the client side: crytographic material can be securely stored on servers and recovered with the help of the OPRF. On the other hand, obliviousness of the PRF evaluation allows to hide client's protocol input. Together with being efficient and strongly secure, these features have made OPRFs one of the most promising privacy-enhancing tools in recent years.

Applications of OPRFs in the literature leverage OPRFs in essentially two ways. First, OPRFs are used to let clients *(re-)compute high-entropy cryptographic objects*, such as cryptographic keys, from her data. Applications include but are not limited to

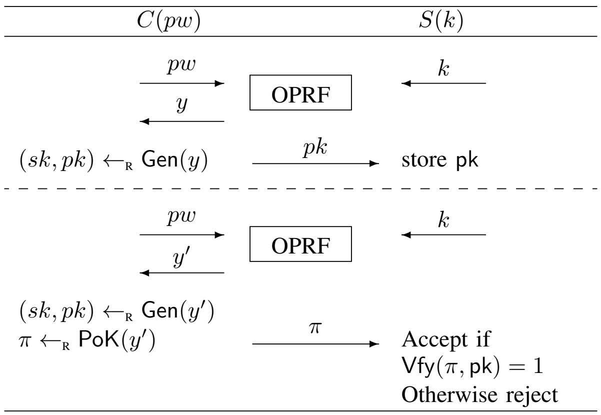
- Secure password verification
- Server-assisted encryption
- Secret key recovery/password-encrypted backups

Second, OPRFs are deployed instead of hash functions, to *enforce interaction when computing hash values*. This is useful in settings where limitation of hash evaluation is desirable. Examples are:

- Precomputation-resistant password hardening
- Rate limiting for web-services
- Secure comparison of private inputs (e.g., contact tracing)

We now detail each of these applications.

Secure password verification. On the internet, password verification is usually deployed through "Password-over-TLS", a mechanism that requires the client to send her clear-text password to the server, who then hashes it and compares the hash against a local database. Using OPRFs, one can implement lightweight protocols to *interactively* compute the hash and have the client prove knowledge of the correct hash afterwards. Crucially, the server does not see the client's password by the obliviousness of the OPRF. We depict a blueprint for secure password verification using an OPRF in Figure 11.



$C(pw)$
$S(k)$
pw
y
OPRF
k
$(sk, pk) \leftarrow_R \mathsf{Gen}(y)$
pk
store pk
pw
y'
OPRF
k
$(sk, pk) \leftarrow_R \mathsf{Gen}(y')$
$\pi \leftarrow_R \mathsf{PoK}(y')$
π
Accept if
$\mathsf{Vfy}(\pi, \mathsf{pk}) = 1$
Otherwise reject

Figure 11: Blueprint for secure password verification from multi-point OPRFs, with registration above the dashed line, and verification below.

Upon registration, the client runs the OPRF protocol to compute a secret key from her password. The server

5. For the sake of consistency with the technical sections, we use the term "client" instead of the more common term "user" also in this section.

PRF	Reference	Model	Security model	Adv. model C-S	Hardness ass.	Description	Modular blocks	Rounds	Comp. costs	Comm. costs
Naor-Reingold (NR)	FIPR05, 1st [4]	◔	SC	● - ◐	DDH	Protocol of Fig. 1	$\binom{2}{1}$-OT for semi-h. S	1	n $\binom{2}{1}$-OTs, S: 1 exp, C: 1 exp	n $\binom{2}{1}$-OTs S $\rightarrow$ C: 1 ge
	HL08 [5]	◔	SC	● - ●	DDH	Protocol of Fig. 1	Maliciously secure $\binom{2}{1}$-OT	1	same as above	same as above
	Hazay15 [26]	◔	SC	● - ●	DDH	Variant of Fig. 2, forced same key, for special NR variant	Hom. enc. (ElGamal), 6 types of ZKP	2	see paper	see paper
	JKK14 [9]	◐	UC	● - ●	Strong RSA, DCR	Variant of Fig. 2 with proofs	Σ-ZKP, 1-time-Sign., Commitment scheme, Hom. encryption	2	$O(n \cdot \lambda)$	mainly 5 Σ-ZKP + $2n$ ctxt (each way)
Dodis-Y. (DY)	JL09 [7]	◔	SC	● - ●	factoring, n-DDHI	Variant of Fig. 4, with 2 public keys and proofs	Verif. & Hom. encryption (Camenish-Shoup [67]), ZKPoK, safe RSA modulus	2	C: 1 π/enc/dec, 2 vfy/exp S: 1 exp/enc/dec/i/vfy, 2 π	C $\rightarrow$ S: 1 π/pk, 2 ctxt S $\rightarrow$ C: 1 ge/pk 2, π/ctxt
	CL17 [22]	◐	UC	● - ◐	q-BDDHI, DCR	Protocol of Fig. 4, with proofs	Server-side Authentication, Certificate Auth., Com., + same as above	1	C: 1 enc/com/π, 2 exp S: 1 vfy/dec/exp	C $\rightarrow$ S: 1 com/ctxt/π S $\rightarrow$ C: 1 ge
	TCRSTW21 [31]	◕	G	● - ●	OM-gapSDHI	Blinded exponentation of DY variant	NIZK of dlog	1	C: 2 exp, 1 vfy S: 1 exp/π	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 1 π/ge
Hashed Diffie-Hellman	JL10 [8]	◕	G	● - ●	OM-gapCDH	2HashDH	—	1	C: 2 exp S: 1 exp	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 1 ge
	JKK14 [9]	◕	UC	● - ●	OM-gapCDH	2HashDH with proof	NIZK of dlog	1	C: 2 exp, 1 v S: 1 exp, 1 π	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 2 ge, 1 π
							DDH-Oracle $\mathcal{O}$ (e.g., pairing)	1	C: 2 exp, 1 $\mathcal{O}$ S: 1 exp	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 2 ge
					OM-RSA	2HashRSA (Fig. 15)	–	1	C: 3 exp S: 1 exp	C $\rightarrow$ S: 2 ge S $\rightarrow$ C: 1 ge
	ECSJR15 [19]	◐	G	● - ●	OM-BDDH	HashDH (Fig. 3), w. pairing & proofs	NIZK of dlog	1	C: 2 exp, 1 v S: 2 exp, 1 π, 1 prg	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 2 ge, 1 π
						threshold version	”	”	”	” for each server
	JKKX16 [10] and JKX18 [14]	◕	UC	● - ●	OM-gapCDH	2HashDH, non-verifiable	Authenticated Channel	1	C: 2 exp S: 1 exp	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 1 ge
	JKKX17 [11]	◕	UC	● - ●	T-OM-gapCDH	2HashDH with multiple servers	Secure & auth. channels	1	C: 2 exp every S: 1 exp	C $\rightarrow$ every S: 1 ge every S $\rightarrow$ C: 1 ge
	DGSTV18 [28]	◕	UC	● - ●	OM-gapCDH	JKK14 [9] with forced same key	NIZK of dlog	1	C: 1 vfy, 2 exp S: 1 exp/π	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 1 ge/π
	BFHLY19 [12]	◕	UC	● - ●	OM-gapBCDH	Distributed 2HashDH, with pairings and proactive security	—	1	C: 2 exp every S: 1 exp/prg	C $\rightarrow$ every S: 1 ge every S $\rightarrow$ C: 1 ge
	Lehmann19 [57]	◕	G	● - ●	OM-DDH-io	3-party HashDH with C=(Req,Rec)	Hom. encryption (ElGamal)	1	Req: 1 enc. S: 1 rand, 1 exp, Rec: 1 dec	Req $\rightarrow$ S: 1 ctxt S $\rightarrow$ Rec: 1 ctxt Req $\rightarrow$ Rec: 1 pk
	DHL22 [13]	◕	UC	◐ - ◐	OM-gapBCDH	Distributed 2HashDH, w. pairings and extendable input	—	1	C: 2 exp S: 1 exp/prg	C $\rightarrow$ S: 1 ge S $\rightarrow$ C: 1 ge

TABLE 4: Algebraic OPRFs and their costs for one evaluation. We omit negligible factors from cost columns, such as transmission of identifiers, sampling, group operations and symmetric operations. Elements in modular building blocks may add to hardness assumptions, model and rounds, and they need to be instantiated to derive final costs. We mark in red OPRFs that are not maliciously secure, or require more than one round. Green marks optimal costs for one evaluation.

Model
◔ = standard (plain)
◐ = CRS
◕ = ROM

Adv. Model
◐ = semi-honest
● = malicious

Security model
SC = secure computation
UC = universal composability
G = game-based

Costs
n = input length
C = client
S = server
exp = exponentiation
mexp = multi-exp
p = pairing

enc = encryption
dec = decryption
rand = randomization
π = proof
prg = pairing
vfy = verify
ge = group element
σ = signature
pk = public key
ctxt = ciphertext
com = commitment

Hardness assumptions
CDH = computational Diffie-Hellman
DDH = decisional Diffie-Hellman
B = bilinear
I = inversion
S = strong
OM = one-more
gap = decision oracle
io = inversion oracle
T = threshold
DCR = dec. composite residuosity
n- = interactive n-type assumption

stores the corresponding public key. To verify the client's password later, the OPRF step is repeated, and the client performs a proof of knowledge of the secret key. The server is convinced that the client is eligible, i.e., knows either the password or the secret key, if the proof verifies. Examples from the literature that use this protocol layout include the asymmetric password-authenticated key exchange (PAKE) protocol OPAQUE [14], the distributed SSO protocol PESTO [12], and the distributed password-authenticated symmetric encryption service DPaSE [13]. Proofs of knowledge can be efficiently implemented by signing a server's nonce [12], [13], or by performing an authenticated key agreement [14]. Recent activities have aimed at integrating such strongly secure password verification into TLS[6].

Secure password verification is usually deployed with Hashed DH OPRFs (Section 2.2), since they have lowest costs when only single evaluations are required, and they offer various advanced properties (see Table 3). Deploying a threshold or distributed OPRF further protects against offline attacks upon server breach [68][7], since a certain number of servers need to be corrupted before they can jointly evaluate the PRF. Partial obliviousness can be used to bind the stored public key to a username, by letting the username be the public part of the input. This allows the server to limit the number of attempts per account.

Server-assisted encryption. Using the OPRF, the client turns data such as her password, a file [16], a file identifier [15], or even a combination thereof [13] into a symmetric key $\mathsf{PRF}_k(\mathtt{data})$, with which she subsequently encrypts her messages, or her hard drive. The server-contributed part of the encryption key, namely the OPRF key k, can be distributed to increase security [13], and can even be made updateable [15]. If the key is derived from the data to be encrypted, then the deterministic nature of the OPRF even allows for deduplication [16], [69].

Secret key recovery. Another use case in line with the two former ones is recovery of cryptographic material from only a password [10], [11], [70]. This is useful to, e.g., rescue encrypted data after losing one's key material, or to recover a cryptocurrency wallet after losing the corresponding secret key. We note that most works on secret key recovery in fact deploy a cryptographic primitive called *password-protected secret sharing* (PPSS), which is in turn most efficiently built from OPRFs [10], [11] (see Appendix B.3 for details). Recently, WhatsApp has rolled out an encryption feature for their chat backup function.[8] Here, the user can restore the encryption key from a password through the PAKE protocol OPAQUE [14], which uses the HashDH PRF.

Precomputation-resistant password hardening. Conventional password hardening functions, such as scrypt or Argon2, can be computed locally and are thus prone to precomputation attacks. Using an interactive OPRF instead of a hardening function prevents such attacks (as long as the key is stored on a different server than the hardened password database). The main idea is that *hard* computation is substituted by *interactive* computation, where the latter limits attacks even more as it requires active participation of the server. A hardening service deployed with an OPRF can even support efficient key updates [19]. The principle works also if password hardening is part of another primitive; for example, plugging an OPRF instead of a hardening function protects password-authenticated key exchange (PAKE) schemes against precomputation attacks [14].

Rate-limiting for web-services. Rate limiting access to web services, a common way to defend against Denial-of-Service attacks, can be enforced by giving out a limited number of access tokens to clients. Interactively computing access tokens of the form $(n, \mathsf{PRF}_k(n))$ with an OPRF, for nonce n chosen by the client, has been demonstrated superior to approaches using signatures [28]. An implementation example is the PrivacyPass protocol[9].

Secure comparison of private inputs. An OPRF can be seen as a protocol for interactively and obliviously evaluating a hash function. This makes OPRFs extremely useful for applications such as private set intersection (PSI) [5] (see Appendix B.4), pattern matching [5], [17], oblivious keyword search (KS) [4], [71] and contact tracing [61], [63] which merely need to compare client inputs but no longer need to compute anything with them. The interactive nature of the OPRF adds significantly to the security level: whenever hashes of private data are shared or even broadcasted (as done in, e.g., Apple's AirDrop protocol), computation of the hash through an OPRF significantly limits the number of brute-force attempts to recover the private data [72]. We exemplarily depict the transition from hash functions to OPRFs for KS [4] in Figure 12.

6. Open Problems and Future Work

We summarize relevant open problems and identify possible future research directions related to ORPFs.

OPRF constructions. In Section 3 we gave an overview of which (combined) properties we can achieve from which constructions. But is there any way to, e.g., batch evaluate OPRFs based on NY or DY, potentially exploring techniques from cryptographic accumulators? Can OPRFs based on NR be distributed? A threshold version for the distributed OPRF of Baum et al. [12] would enhance usability of their SSO scheme.

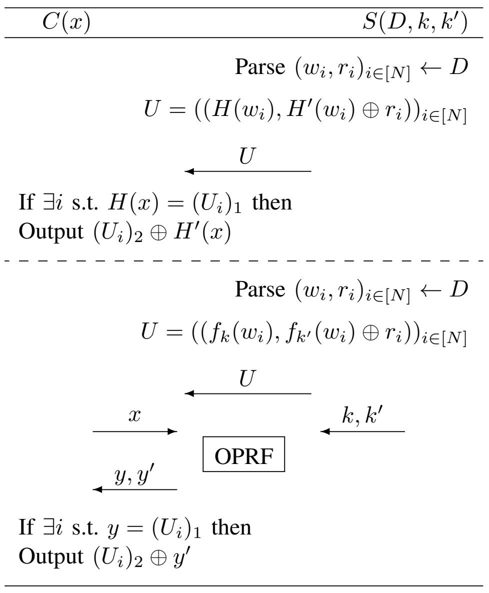
Oblivious weak PRFs. Weak PRFs appear pseudorandom only on randomly chosen inputs and are a strictly weaker primitive than PRFs. There exist many efficient constructions, with an interesting recent line of research on LPN-based weak PRFs [73], [74]. Weak PRFs can be more efficient than PRFs, and they find applications in symmetric primitives such as encryption or message authentication codes, and can be even used to construct signature schemes. It is an interesting open question whether weak PRFs can be evaluated obliviously, how to even define security of such a primitive, and what applications they can be used for.

6. https://blog.cloudflare.com/opaque-oblivious-passwords/

7. The concept of a distributed, verifiable, and proactively secure OPRF appears only implicitly in [68].

8. https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf

9. https://asecuritysite.com/encryption/privacypass



$C(x)$
$S(D, k, k')$
Parse $(w_i, r_i)_{i\in[N]} \leftarrow D$
$U = ((H(w_i), H'(w_i) \oplus r_i))_{i\in[N]}$
U
If $\exists i$ s.t. $H(x) = (U_i)_1$ then
Output $(U_i)_2 \oplus H'(x)$
Parse $(w_i, r_i)_{i\in[N]} \leftarrow D$
$U = ((f_k(w_i), f_{k'}(w_i) \oplus r_i))_{i\in[N]}$
U
x
k, k'
OPRF
y, y'
If $\exists i$ s.t. $y = (U_i)_1$ then
Output $(U_i)_2 \oplus y'$

Figure 12: **Top:** "Insecure" KS for searching keyword x in an N-sized database D, using hash functions H, H'. $(U_i)_1$, e.g., denotes the first entry in the i-th pair of list U. The client can reveal the whole database by exhaustive search of keywords w_i. **Bottom:** Oblivious KS [4], substituting hashes with OPRF evaluations of PRF f. Client C can reveal at most one database entry r_i, and the server does not learn which one.

Dealing with malicious servers and weak keys. The oblivious nature of OPRFs makes it sometimes challenging to enforce correctness of the client's output. Many OPRFs rely on expensive zero knowledge proofs to enforce correct behavior of the server (see Table 4 for references), but there exist cases in which not even these techniques can be applied, and no (reasonably efficient, single-evaluation) protocol is known that works for a malicious server [4], [5], [13]. Moreover, most OPRF constructions let the client check honest server behavior by verifying proofs w.r.t a public key generated by the server. This leaves OPRFs and their implementation to the following pitfall: if the server chooses (intentionally or not) a non-uniform key, then from the perspective of the client all checks pass, and she might now, e.g., encrypt her data with a weak key. Integrating well-formedness of keys into OPRFs has not yet been considered, but would significantly improve security guarantees of many applications.

Unification of security notions. Despite significant time spent researching, even the authors of this work lost track of the number of UC functionalities for OPRFs. Unification efforts seem required to streamline existing definitions into one that captures all necessary aspects, but is still strong enough for the respective applications.

Efficiency of OT-based OPRFs. The original ORPF construction in [4] requires n parallel OT instances (see Table 4), which with the best OT techniques translates to $O(n)$ exponentiations. OT is one of the best researched primitives in cryptography, and new variants and efficiency improvements arise almost on a weekly basis. Since OT and OPRFs are strongly related (see Figure 13), OPRFs can directly benefit from OT improvements. It is thus important to "keep up" with OT research from the side of OPRFs. For example, it is still not known how to exploit one of the biggest advancements in OT efficiency, namely OT Extensions [39], to improve efficiency of the NR-based OPRFs evaluated with n $\binom{2}{1}$-OTs. Another example is a recent OT variant called *silent OT* [74], which is claimed to yield a batched OPRF with costs as little as 1 bit of communication per OPRF evaluation on a random input, while computational costs increase. But can silent OT only be applied to generic OPRF evaluation, or can it also improve, e.g., oblivious evaluation of the NR PRF? Can it be applied to improve efficiency of the PSI-tailored batched OPRFs shown in Table 5?

Adaptation in practice. To put forth usage of OPRFs, they need to be standardized and integrated into existing protocols such as, e.g., TLS 1.3. The Internet Research Task Force (IRTF) has recently recommended[10] usage of the PAKE protocol OPAQUE, which protects against precomputation attacks by using an OPRF. Currently, the IRTF supports drafting of an RFC [23] for the verifiable Hashed DH OPRF of Jarecki et al. [9]. Some examples of upcoming IRTF RFCs with protocols deploying OPRFs are OPAQUE [24] and its integration into TLS 1.3, called TLS-OPAQUE [25]. The latter protocol is a potential candidate for replacing the ubiquitous but completely non-private "password-over-TLS" mechanism. All these works need to be driven forward by a joint effort of the community.

Quantum-secure ORPFs. With the exception of OPRFs based on symmetric primitives, all known efficient OPRF constructions rely on discrete-log- or factoring-type hardness assumptions. These assumptions are known to fall with the rise of quantum computers. While it is true that some of the primitives that act as building blocks of OPRF constructions (such as OT – one can instantiate Yao's garbled circuit protocol with a post-quantum-secure OT, which can then be used to obliviously evaluate an AES cricuit) can be instantiated quantum-securely, there is only little work so far in designing special-purpose and efficient quantum-secure algorithms for OPRFs. The two notable exceptions are the lattice-based OPRF by Albrecht et al. [20], and the isogeny-based OPRF by Boneh et al. [21]. Both constructions constitute nice feasibility results, and more research is needed to improve their efficiency. Moreover, neither of the two OPRFs comes with any of the properties mentioned throughout this paper.

References

[1] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, vol. 33, no. 4, pp. 792–807, 1986.

[2] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM (JACM)*, vol. 51, no. 2, pp. 231–262, 2004.

[3] ——, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM (JACM)*, vol. 51, no. 2, pp. 231–262, 2004.

[4] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography Conference*. Springer, 2005, pp. 303–324.

10. https://github.com/cfrg/pake-selection

[5] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Theory of Cryptography Conference*. Springer, 2008, pp. 155–175.

[6] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 818–829.

[7] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection," in *Theory of Cryptography Conference*. Springer, 2009, pp. 577–594.

[8] ——, "Fast secure computation of set intersection," in *International Conference on Security and Cryptography for Networks*. Springer, 2010, pp. 418–435.

[9] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and T-PAKE in the password-only model," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 233–253.

[10] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011, pp. 433–444.

[11] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "TOPPSS: cost-minimal password-protected secret sharing based on threshold OPRF," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 39–58.

[12] C. Baum, T. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai, "PESTO: proactively secure distributed single sign-on, or how to trust a hacked server," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2020, pp. 587–606.

[13] P. Das, J. Hesse, and A. Lehmann, "DPaSE: Distributed Password-Authenticated Symmetric Encryption," *Cryptology ePrint Archive*, 2020.

[14] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 456–486.

[15] S. Jarecki, H. Krawczyk, and J. Resch, "Updatable oblivious key management for storage systems," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 379–393.

[16] J. Camenisch, A. D. Caro, E. Ghosh, and A. Sorniotti, "Oblivious PRF on committed vector inputs and application to deduplication of encrypted data," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 337–356.

[17] S. Faust, C. Hazay, and D. Venturi, "Outsourced pattern matching," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2013, pp. 545–556.

[18] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1447–1464.

[19] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, "The Pythia PRF service," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 547–562.

[20] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart, "Round-optimal verifiable oblivious pseudorandom functions from ideal lattices," in *IACR International Conference on Public-Key Cryptography*. Springer, 2021, pp. 261–289.

[21] D. Boneh, D. Kogan, and K. Woo, "Oblivious pseudorandom functions from isogenies," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 520–550.

[22] J. Camenisch and A. Lehmann, "Privacy-preserving user-auditable pseudonym systems," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 269–284.

[23] A. Davidson, N. Sullivan, and C. Wood, "Oblivious pseudorandom functions (OPRFs) using prime-order groups," https://datatracker.ietf.org/doc/draft-irtf-cfrg-voprf/, Internet Engineering Task Force, Tech. Rep., 2019.

[24] H. Krawczyk, D. Bourdrez, K. Lewi, and C. Wood, "The OPAQUE asymmetric PAKE protocol," https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/, 2021.

[25] N. Sullivan, D. Krawczyk, O. Friel, and R. Barnes, "OPAQUE with TLS 1.3," https://datatracker.ietf.org/doc/draft-sullivan-tls-opaque/, Internet Engineering Task Force, Tech. Rep., 2021.

[26] C. Hazay, "Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs," *Journal of Cryptology*, vol. 31, no. 2, pp. 537–586, 2018.

[27] M. Abdalla, M. Cornejo, A. Nitulescu, and D. Pointcheval, "Robust password-protected secret sharing," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 61–79.

[28] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 164–180, 2018.

[29] S. Jarecki, H. Krawczyk, and J. Xu, "On the (In) Security of the Diffie-Hellman oblivious PRF with multiplicative blinding," in *IACR International Conference on Public-Key Cryptography*. Springer, 2021, pp. 380–409.

[30] P. Miao, S. Patel, M. Raykova, K. Seth, and M. Yung, "Two-sided malicious security for private intersection-sum with cardinality," in *Annual International Cryptology Conference*. Springer, 2020, pp. 3–33.

[31] N. Tyagi, T. Ristenpart, N. Sullivan, S. Tessaro, C. A. Wood *et al.*, "A fast and simple partially oblivious PRF, with applications," *Cryptology ePrint Archive*, 2021.

[32] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 250–267.

[33] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications." *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 177–197, 2017.

[34] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1257–1272.

[35] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 122–153.

[36] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "SpOT-light: lightweight private set intersection from sparse OT extension," in *Annual International Cryptology Conference*. Springer, 2019, pp. 401–431.

[37] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing*, 1999, pp. 245–254.

[38] ——, "Oblivious transfer with adaptive queries," in *Annual International Cryptology Conference*. Springer, 1999, pp. 573–590.

[39] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Annual International Cryptology Conference*. Springer, 2003, pp. 145–161.

[40] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[41] C. Hazay, "Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs," *Journal of Cryptology*, vol. 31, no. 2, pp. 537–586, 2018.

[42] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Annual Cryptology Conference*. Springer, 2011, pp. 111–131.

[43] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and KDCs," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 327–346.

[44] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*. Springer, 1983, pp. 199–203.

[45] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.

[46] W. Ford and B. S. Kaliski, "Server-assisted generation of a strong secret from a password," in *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*. IEEE, 2000, pp. 176–180.

[47] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Annual International Cryptology Conference*. Springer, 1992, pp. 89–105.

[48] B. A. Huberman, M. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999, pp. 78–86.

[49] R. Agrawal, A. Evfimievski, and R. Srikant, "Information sharing across private databases," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003, pp. 86–97.

[50] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *International Workshop on Public Key Cryptography*. Springer, 2005, pp. 416–431.

[51] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Annual International Cryptology Conference*. Springer, 2004, pp. 41–55.

[52] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *Annual International Cryptology Conference*. Springer, 2009, pp. 108–125.

[53] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167.

[54] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 2, pp. 1–35, 2018.

[55] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 430–454.

[56] M. Chase and P. Miao, "Private set intersection in the internet setting from lightweight oblivious PRF," in *Annual International Cryptology Conference*. Springer, 2020, pp. 34–63.

[57] A. Lehmann, "ScrambleDB: Oblivious (Chameleon) Pseudonymization-as-a-Service." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 289–309, 2019.

[58] S. Jarecki, H. Krawczyk, and J. Resch, "Threshold partially-oblivious PRFs with applications to key management," *Cryptology ePrint Archive*, 2018.

[59] R. Cramer, I. Damgård, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," in *Theory of Cryptography Conference*. Springer, 2005, pp. 342–362.

[60] B. Hemenway Falk, D. Noble, and R. Ostrovsky, "Private set intersection with linear communication from general assumptions," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 14–25.

[61] T. Duong, D. H. Phan, and N. Trieu, "Catalic: Delegated PSI cardinality with applications to contact tracing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 870–899.

[62] N. Chandran, D. Gupta, and A. Shah, "Circuit-PSI with linear complexity via relaxed batch OPPRF," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 353–372, 2022.

[63] P. Rindal and P. Schoppmann, "VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 901–930.

[64] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 515–530.

[65] Y. Lindell, "A tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, pp. 277–346, 2017.

[66] M. Abe and M. Ohkubo, "A framework for universally composable non-committing blind signatures," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 435–450.

[67] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Annual International Cryptology Conference*. Springer, 2003, pp. 126–144.

[68] J. Camenisch, A. Lehmann, and G. Neven, "Optimal distributed password verification," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 182–194.

[69] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Annual International Conference on the Theory and Applications of Cryptographic techniques*. Springer, 2013, pp. 296–312.

[70] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven, "Memento: How to reconstruct your secrets from a single password in a hostile environment," in *Annual Cryptology Conference*. Springer, 2014, pp. 256–275.

[71] W. Ogata and K. Kurosawa, "Oblivious keyword search," *Journal of Complexity*, vol. 20, no. 2-3, pp. 356–371, 2004.

[72] A. Heinrich, M. Hollick, T. Schneider, M. Stute, and C. Weinert, "PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3577–3594.

[73] D. Boneh, Y. Ishai, A. Passelègue, A. Sahai, and D. J. Wu, "Exploring crypto dark matter," in *Theory of Cryptography Conference*. Springer, 2018, pp. 699–729.

[74] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent OT extension and more," in *Annual International Cryptology Conference*. Springer, 2019, pp. 489–518.

[75] A. Bogdanov and A. Rosen, "Pseudorandom functions: Three decades later," in *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 79–158.

[76] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, vol. 28, no. 6, pp. 637–647, 1985.

[77] G. Brassard and C. Crépeau, "Oblivious transfers and privacy amplification," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1997, pp. 334–347.

[78] J. Camenisch, G. Neven *et al.*, "Simulatable adaptive oblivious transfer," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 573–590.

[79] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 1–19.

[80] C. Hazay and Y. Lindell, "Efficient oblivious polynomial evaluation with simulation-based security," *Cryptology ePrint Archive*, 2009.

[81] M. O. Rabin, "How to exchange secrets with oblivious transfer," *Cryptology ePrint Archive*, 2005.

[82] J. Kilian, "Founding cryptography on oblivious transfer," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988, pp. 20–31.

[83] G. Brassard, C. Crépeau, and J.-M. Robert, "All-or-nothing disclosure of secrets," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 234–238.

[84] C. Crépeau and G. Savvides, "Optimal reductions between oblivious transfers using interactive hashing," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 201–221.

[85] R. Impagliazzo and S. Rudich, "Limits on the provable consequences of one-way permutations," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, 1989, pp. 44–61.

[86] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Annual Cryptology Conference*. Springer, 2013, pp. 54–70.

[87] M. Orrù, E. Orsini, and P. Scholl, "Actively secure 1-out-of-N OT extension with application to private set intersection," in *Cryptographers' Track at the RSA Conference*. Springer, 2017, pp. 381–396.

[88] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 797–812.

[89] D. Malkhi and Y. Sella, "Oblivious transfer based on blind signatures," Technical Report 2003-31, Leibniz Center, Hebrew University, Tech. Rep., 2003.

[90] M. Fischlin, "Round-optimal composable blind signatures in the common reference string model," in *Annual International Cryptology Conference*. Springer, 2006, pp. 60–77.

[91] A. Kiayias and H.-S. Zhou, "Equivocal blind signatures and adaptive UC-security," in *Theory of Cryptography Conference*. Springer, 2008, pp. 340–355.

[92] C.-K. Chu and W.-G. Tzeng, "Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries," in *International Workshop on Public Key Cryptography*. Springer, 2005, pp. 172–183.

[93] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme," in *International Conference on Financial Cryptography*. Springer, 2001, pp. 319–338.

[94] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 514–532.

[95] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011, pp. 433–444.

[96] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *1986 IEEE Symposium on Security and Privacy*. IEEE, 1986, pp. 134–134.

[97] E. De Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear computational and bandwidth complexity," *Cryptology ePrint Archive*, 2009.

Appendix A. A non-technical Introduction to OPRFs

Hash functions are ubiquitous in the digital world. They map arbitrary inputs to a bitstring of fixed length. The resulting hash looks like a random bit sequence and it does not reveal which input it was computed from.

In practice, such hash functions are used to verify integrity of data via checksums, to demonstrate knowledge of hashed values, or to protect data such as, e.g., passwords. In some of these applications, it is desirable to additionally equip the hash function $H()$ with a seed, i.e., compute

$$H(s||x).$$

The seed s might be a public nonce, which is useful for applications where it is crucial to ensure freshness of the hash. If on the other side the seed is kept secret, it plays the role of a secret key: only the owner of the secret key can compute the hash value. A real-world example with secret seeds are password authentication protocols, where it is common practice to not store plaintext passwords pw but hashes $H(s, pw)$ of them with a seed s that the authentication server ideally should keep secret in order to prevent precomputation attacks.

This can be seen as some form of keying a hash function. Let's go one step further and make hash computation interactive: imagine not only the server (who knows the seed s) can compute hash values, but also clients can compute them with the help of the server. A trivial such protocol would be the following.

Client holds pw		Server holds s
	$\xrightarrow{pw}$	$h \leftarrow H(s\|\|pw)$
output h	$\xleftarrow{h}$	

From the perspective of the server S, the protocol has the nice property of preventing the client from learning seed s. However, privacy of the client is not protected well, since S learns the value h computed by the client, and even the input pw. Can we improve this?

Oblivious pseudorandom functions not only answer this question in the affirmative. They provide protection of the client's privacy. Namely, they let clients compute keyed hash functions *interactively* with a server who holds the key, without the client learning the key, and without the server learning the client's input or the hash value.

While it is instructive to think about OPRFs in terms of interactive keyed hashing, their formalization uses the related concept of a *pseudorandom function* (PRF). In a nuthsell, a PRF is a keyed function that behaves essentially like a random function. We give a formal definition of PRFs below.

We recommend the reader to continue with Section 2, to learn how to construct perfectly private alternatives of the small protocol above, namely how to construct an *oblivious pseudorandom function*. Readers wondering how interactive and keyed hashing is useful directly look at Section 5.

Pseudorandom functions have been introduced by Goldreich, Goldwasser and Micali [1]. We give a formal definition below and also point the interested reader to a good survey [75].

Definition 6 (Pseudorandom function)**.** A family of functions $f_k : \{0,1\}^m \to \{0,1\}^n$ with key $k \in \{0,1\}^\lambda$ is called *pseudorandom* if the following holds.

- $f_k(x)$ is efficiently computable from k and x.
- It is not efficiently decidable whether one has access to a computation oracle for $F_s(\cdot)$ or to an oracle producing random bitstrings of length n.

Appendix B. Relations to Other Cryptographic Primitives

In this section, we put OPRFs in context with related cryptographic primitives. We provide a comprehensive overview in Figure 13, and discuss the most prominent relations in the following.

B.1. Oblivious Transfer

Oblivious transfer (OT) ("Rabin-OT", [81]) is an oblivious channel between two parties with erasure probability 1/2. It was demonstrated to be sufficient for

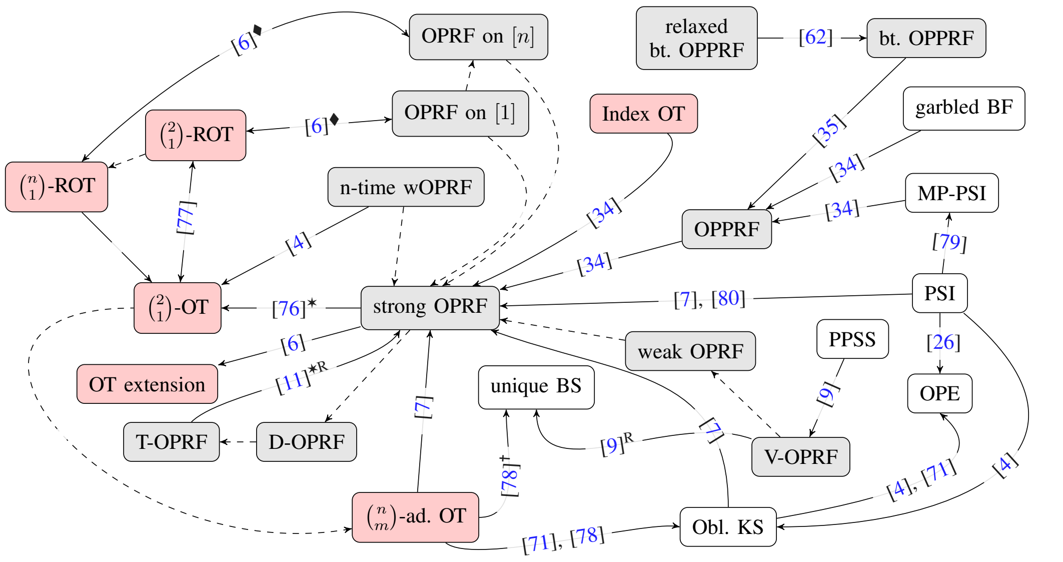


Figure 13: Reductions between OPRFs (gray) and related cryptographic primitives, in particular OT variants (red).

⟶ reduces to
⇢ trivial
✶ = generic SFE
† = under additional assumptions
♦ = non-generic
R = Random oracle model

wOPRF = weak OPRF
BS = Blind signatures
OT = Oblivious transfer
ROT = Random OT
KS = Oblivious keyword search
PSI = Private set intersection

MP = Multi-party
OPE = Obl. polynomial evaluation
OPPRF = Obl. programmable PRF
BF = Bloom filters
bt. = batched
ad. = adaptive

any 2-party secure function evaluation (SFE) by Kilian [82]. Even et al. [76] later introduced the information-theoretically equivalent version of OT that we are most familiar with today: in 1-out-of-2 OT ($\binom{2}{1}$-OT), Alice provides two bits m_0, m_1, and Bob provides a bit b. Alice learns nothing, and Bob learns only m_b. In other words, $\binom{2}{1}$-OT is a secure computation of $\mathcal{F}_{OT} : ((m_0, m_1), b) \to (\bot, m_b)$. The concept generalizes from bits to k-bit messages, denoted $\binom{2}{1}$-OT^k, and further to $\binom{n}{1}$-OT^k [83] and (static or adaptive) $\binom{n}{m}$-OT^k [38]. We can also consider OT of random messages (ROT) [84], OT extension [39] and batching OT [5].

The close relation between OT and OPRFs becomes apparent with the following construction of $\binom{2}{1}$-OT^k from one execution of a strong OPRF (Def. 1) evaluating PRF f with key space $\mathcal{K}$ and domain 2^k.

- Alice, on input m_0, m_1 chooses $k \leftarrow_R \mathcal{K}$, computes $c_0 \leftarrow m_0 \oplus f_k(0)$, $c_1 \leftarrow m_1 \oplus f_k(1)$ and sends c_0, c_1 to Bob.
- Alice and Bob engage in one execution of the OPRF protocol, with Alice playing the role of the server with input k, and Bob in the role of the client with input $b \in \{0, 1\}$. Bob learns $f_k(b)$, Alice learns nothing.
- Bob decrypts $m_b \leftarrow c_b \oplus f_k(b)$ and outputs m_b.

The construction generalizes as depicted in Figure 14 to adaptive $\binom{n}{m}$-OT^k from m subsequent evaluations of the strong OPRF [7]. The fact that such a strong variant of OT efficiently reduces to strong OPRFs demonstrates the power of OPRFs as cryptographic primitive, and yields constructions of OT variants such as $\binom{n}{m}$-ROT^k via the known reduction from $\binom{n}{m}$-OT^k to ROT [84].

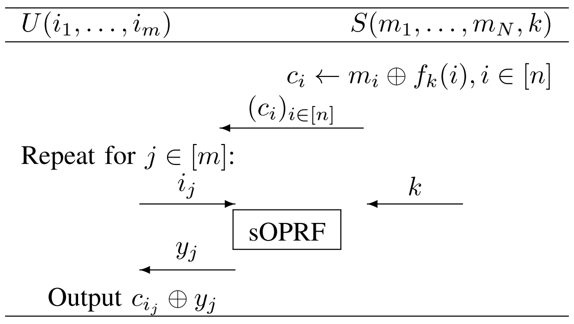


Figure 14: Transformation from a strong OPRF to adaptive $\binom{n}{m}$-OT^k. The client can input its indices in an arbitrary order and hence retrieve t evaluations in an adaptive fashion.

B.1.1. OT of random messages. In random OT (ROT), the client learns random m_0, m_1, while the receiver learns m_b for a choice bit $b \in \{0, 1\}$. Thus, the messages m_0, m_1 are uniformly generated by the protocol itself (note the similarities between this definition and the definition of single-point OPRF from Section 3.5, where the server obtains a PRF key uniformly generated by the OPRF protocol).

One execution of random OT corresponds to obliviously evaluating the PRF $f_{(m_0,m_1)}(r) := m_r$ with input domain $\{0, 1\}$ [6]. More generally, we can interpret a 1-out-of-n OT of random messages as an oblivious evaluation of the above PRF with input domain $\{1, \dots, n\}$.

B.1.2. OT Extension. While OTs are highly unlikely to be possible without using public key cryptography [85], it was shown that it is possible to extend a small number λ

of "base" OTs to a large (i.e., polynomial in λ) number of OTs using only symmetric key primitives. Such a protocol is called an *OT extension* [39], [86]. Kolesnikov et al. [6] were the first to observe that the extension protocol of Ishai et al. [39] can be interpreted as a weak single-point ORPF protocol, and this analogy was further developed in [36], [54], [87], among others. Constructions of single-point OPRFs from OT extension where made explicit in several works [6], [54], [88]. Pinkas et al. [36] even construct a multi-point OPRF from OT extension, but such that the client must choose all the query points before the OPRF key is generated (this corresponds to the OPPRF functionality introduced by Kolesnikov et al. [34]).

B.2. Blind Signatures

Blind signatures [44] were first proposed as a tool to provide anonymity in E-cash systems. A blind signature scheme is an interactive protocol involving a client and an authority, where the client can get a message signed by the authority, but without revealing the contents of the message nor the signature. Blind signatures are powerful primitives (see, e.g., [89] for a construction of OT from Chaum's RSA blind signature protocol). The resemblance with OPRFs regarding obliviousness is apparent, but both primitives differ in guarantees (pseudorandomness of OPRF output vs. unforgeability of signatures) and functionality (for verifying a signature we may reveal the message, which we keep secret when verifying OPRF computation). Moreover, signing algorithms may be indeterministic, while OPRFs as function evaluation protocols are inherently deterministic. Nonetheless, the concepts of blind signatures have been a source of inspiration for OPRFs. For example, the DY PRF $f_k^{\mathsf{DY}}(x) = g^{1/(k+x)}$ (Section 2.3) is based on the Boneh-Boyen weak signature defined as $Sign_{sk}(x) = g^{1/(x+sk)}$. Some DY-based OPRF protocols [7], [22] are very similar to particular constructions for blind signatures [52]. UC-secure blind signatures [66], [90], [91] have inspired initial definitions of UC-secure OPRFs [9] as well as constructions: the V-OPRF protocol of Jarecki et al. [9] is obtained by hashing a blind signature-message pair. Moreover, a tweak in UC OPRF definitions called the "ticketing mechanism", which essentially allows for lazy extraction of inputs provided by a corrupted client, is inspired by measures to postpone message extraction in UC definitions of blind signatures [66].

B.2.1. Unique blind signature. We now restrict our attention to unique blind signature schemes [71], [92]. Unfortunately, existential unforgeability of signatures does *not* imply pseudorandomness: take any unforgeable signature scheme and append each signature with a 1; the resulting signatures are still unforgeable, but not pseudorandom. While this rules out a generic reduction from OPRFs to unique blind signature schemes, the two unique blind signature schemes from the literature do in fact produce pseudorandom signatures. Firstly, Chaum [44] proposed a unique blind signature scheme, known as Chaum's RSA construction, which we describe in Figure 15. The figure assumes an RSA setting with $N = pq$, where p, q are two large equal-size primes, and e, d two integers satisfying $ed \equiv 1 \mod \phi(N)$, where $\phi(N) = (p-1)(q-1)$ denotes Euler's phi function. One can notice how the exponential blinding idea behind the modern hashed DH protocols (see Section 2.2) shares the same principle as in Chaum's blind signature scheme. Security for Chaum's blind signature scheme was only proven years later [93] and required introduction of a one-more type variant of RSA. The second unique blind signature scheme from the literature is that of Boldyreva [45], which is DH-based (more concretely, based on the Gap Diffie-Hellman assumption). We depict it in Figure 16.

Both the above schemes are closely related to the modern hashed DH protocols for OPRFs (Section 2.2). Besides all of them using the same principle – namely, double-blinded exponentation – we can even find OPRF protocols that are almost analogous to the corresponding signing procedures. As we show in Figures 15 and 16, Mult-2HashDH [29] is analogous to signing with Boldyreva's scheme, and the 2HashRSA OPRF [9] (ignoring artifacts that stem from UC security and verifiability) is essentially signing with Chaum's scheme.[11]

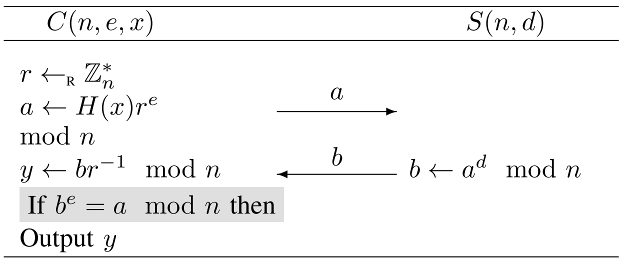

$C(n, e, x)$
$S(n, d)$
$r \leftarrow_R \mathbb{Z}_n^*$
$a \leftarrow H(x)r^e \mod n$
a
b
$b \leftarrow a^d \mod n$
$y \leftarrow br^{-1} \mod n$
If $b^e = a \mod n$ then
Output y

Figure 15: Signing with Chaum's blind RSA signature scheme [44] with message x, and the 2HashRSA OPRF [9]. Both schemes' security proofs [9], [93] require the hash function to be modeled as random oracle.

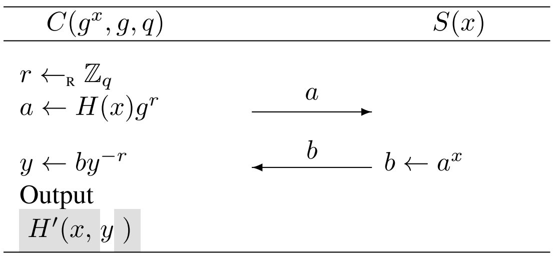


Figure 16: Signing with Boldyreva's blind GDH scheme [45], and the Mult-2HashDH OPRF [29] that multiplicatively blinds the client's input x.

Finally, the short signature scheme of Boneh, Lynn and Shacham [94] can be seen as an implementation of 2HashDH over a bilinear group.

B.3. Password-Protected Secret Sharing

A Password-Protected Secret Sharing (PPSS) [95] scheme is a special secret sharing scheme where Share and Reconstruct procedures take a password as additional input, and correctness of Reconstruct holds only if the

11. We note here that some works claim slightly inaccurately that 2HashDH is an equivalent of Chaum's blind signature.

same password is used as in Share. Jarecki et al. [9] noticed that PPSS can be efficiently built from (single-server) OPRFs, by encrypting every single share with a PRF value derived from the client's password.

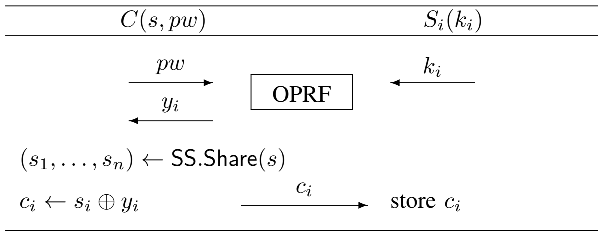

$C(s, pw)$
$S_i(k_i)$
pw
k_i
OPRF
y_i
$(s_1, \dots, s_n) \leftarrow \mathsf{SS.Share}(s)$
$c_i \leftarrow s_i \oplus y_i$
c_i
store c_i

Figure 17: A generic construction of PPSS from verifiable OPRFs, depicting the Share procedure per server S_i. SS.Share denotes a standard secret sharing scheme, such as Shamir.

To protect the client from malicious servers, either the encryption scheme or the OPRF must be verifiable. Instead of running one OPRF with every server in parallel, one can directly use a *threshold* OPRF [11], [68] executed between the client and all servers.

B.4. Private Set Intersection

Private set intersection (PSI) enables two parties P_1 and P_2 with respective input sets X and Y to compute their intersection $X \cap Y$. The different protocols for computing PSI split into two categories: (a) those that compute the intersection itself, and (b) constructions that output $f(X \cap Y)$ for some function f [35] (also known as *circuit-based PSI*). While the first PSI protocols were mostly based on public-key cryptography [3], [96], [97], Hazay et al. [5] initiated a line of PSI research by noticing that PSI can be immediately built from OPRFs. We depict their generic construction in Figure 18.

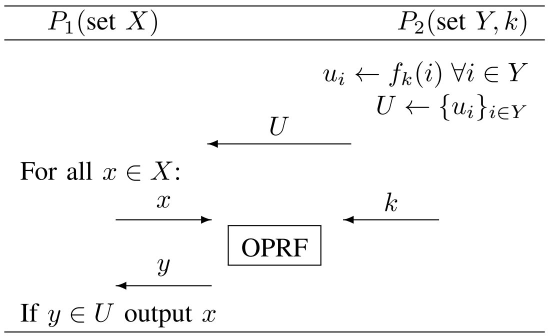


Figure 18: PSI from OPRFs [5], for sets X, Y of potentially different size.

The protocol can be modified to one where both P_1 and P_2 output the intersection set, by having both commit to their inputs and run the protocol in both directions [7]. Besides that, it does not seem to make sense to equip the OPRF with the properties described in Section 3, such as distribution (it is essential that P_2 can evaluate the PRF himself), proactive security (fresh keys can be used for every protocol run anyway), or verifiability (even if P_2 could not cheat in the V-OPRF, he can easily cheat while sending U). Hence, the above PSI blueprint is an example of a protocol where "plain" OPRFs already work well.

The above PSI protocol can be further improved, e.g., by speeding up computation of $f_k(\cdot)$ at P_2, or by applying batching techniques to let P_1 compute the $|X|$ PRF values more efficiently. Efforts to improve efficiency have brought up special OPRF protocols case-tailored for PSI. We now give more details on all these aspects.

Which OPRF is best for PSI? The choice of OPRF protocol depends on the limitations of the setting. Some examples of settings to consider are:

(1) Large difference in set sizes
(2) $|X| \approx |Y|$
(3) Low channel bandwith

One way to speed up computation of P_2 in case of $|Y| \gg |X|$ is to take an OPRF whose underlying PRF can be computed with low effort. Here, PRFs based on symmetric primitives, such as AES encryption, are orders of magnitude faster to compute than NR, DY and HashedDH PRFs, since all of the latter require one exponentiation. An OPRF for $f_k^{\mathsf{AES}}()$ is then given by Yao's Garbled Circuit [53] protocol, and by efficiency improvements thereof [18], [32], [33]. When using such an OPRF to instantiate the protocol from Figure 18, the resulting PSI protocol features low computational costs for P_2, which is beneficial especially when P_2's set is much larger than the one of P_1. If the parties have low bandwith connectivity, Hashed DH OPRFs yield the fastest PSI instantiation, since they have lowest communication costs [33], [54].

In case of $|X| \gg |Y|$, or more generally whenever $|X|$ is large, efficient batching of $|X|$ OPRF executions is an option to speed up the PSI protocol in such settings. Further, Kolesnikov et al. [6] observe that, for the case of $|X| = |Y| =: n$, which can easily achieved by padding if $|X| \approx |Y|$, the PRF evaluations can be performed with respect to *different*[12] PRF keys $k_1, \dots, k_n$. This requires pre-sorting the elements in the set held by the OPRF client, in a way that allows the OPRF server to determine which key was used for which element. In practice, a bin-hashing technique called Cuckoo Hashing is used, and the choice of key used per element can only be narrowed down to 3 keys (which makes comparison less efficient, but significantly speeds up sorting). We refer the reader to [6], Section 5.2, for a nice explanation of this PSI protocol. From the viewpoint of OPRFs, the gist is that PSI for same-sized sets can be implemented with batch evaluation of n OPRFs, allowing arbitrary keys per evaluation. Kolesnikov et al. [6] provide such an OPRF based on any PRF, and efficiency of their construction was improved afterwards [54].

Special-purpose OPRFs for PSI. Recently, *programmable* OPRFs (OPPRF) were introduced by Kolesnikov et al. [34]. An OPPRF is an OPRF protocol for which the server can program a limited number of point-value pairs of the function that is evaluated by the OPRF. The client cannot tell whether she evaluated a programmed or a non-programmed point. Kolesnikov et al. demonstrate that OPPRFs are extremely useful for PSI, even in the multi-party setting. The idea is to let

12. The idea of using different PRF keys is already implicitly contained in the "Circuit-Phashing" PSI protocol of [64].

parties set up OPPRFs that are programmed at $|X|$, to special values that later help figuring out whether all other parties evaluated this OPPRF at some $x \in X$. OPPRFs can be obtained generically from OPRFs [34], or, more efficiently, from OPRFs and linear system solvers [62]. The concept of OPPRFs was extended to batched OPPRFs [35], relaxed OPPRFs (where the client learns few values of which one is the PRF value), and a combination thereof [62]. Further special properties are shuffling of OPRF outputs [30], and secret-sharing of outputs [61]. We note that all these properties are exclusively used by PSI protocols. In some cases, these special OPRFs have "dropped out" of the larger PSI protocols during optimization. Nonetheless, the concepts are related and we hence provide a comprehensive overview of OPRFs for PSI in Table 5.

Reference	**Underlying PRF**	**Special Properties**	**Single-/Multi-point**	**Batching**	**For PSI protocol**	**PSI set sizes**	**Optimized for PSI setting**
HL08 [5]	NR	—	$*$	—	Fig. 18 [5]	any	—
PSSW09 [32]	AES	—	$*$	—	Fig. 18 [5]	any	(1)
JL10 [8]	2HashDH	—	$*$	—	Fig. 18 [5]	any	(3)
KKRT16 [6]	any	related keys	$\bullet$	✔	Circuit-phasing PSI [64]	$=$	(2)
KLSAP17 [33]	AES	—	$*$	—	Fig. 18 [5]	any	(1)
KMPRT17 [34]	any	programmable	$\bullet$	—	Multi-Party PSI from OPPRF [34]	any	—
PSZ18 [54]	any	related keys	$\bullet$	✔	Circuit-phasing PSI [64], optimized with OT Extensions	$=$	(2)
KRSSW19 [18]	LowMC & NR	—	$*$	—	Fig. 18 [5] with correlated random OT precomputation	any	(1)
PRTY19 [36]	any	—	$*$	—	Fig. 18, optimized with hashing and sparse OT extension	any	(3)
PSTY19 [35]	any	programmable	$\bullet$	✔	Circuit PSI, Variant of [64] with OPPRFs	$=$	(2)
DPT20 [61]	any	secret-shared output	$\bullet$	—	Delegated PSI Cardinality [61]	$=$	(1)
CM20 [56]	any	—	$*$	—	Variant of [6] for multi-point OPRFs	any	(3)
MPRSY20 [30]	DY	shuffled	$*$	—	PSI Sum with Cardinality [30]	any	—
CGS21 [62]	any	relaxed, programmable	$\bullet$	✔	Circuit-PSI of [35]	$=$	—
RS21 [63]	any	programmable	$\bullet$	✔	Circuit-PSI of [35]	any	(3)

TABLE 5: OPRFs suitable or even case-tailored for private set intersection. First 4 columns are about the OPRF introduced (or improved, in case of [32], [33]) in the referenced work, the last three columns about the PSI protocol the OPRF is used for. For the OPRF part, $*$ stands for multi-point evaluation, $\bullet$ for single-point. Special properties are briefly described in Appendix B.4 and Section 3.8. For the PSI part, we recall the PSI settings indicated in the last column: (1) $|X| \gg |Y|$ or $|X| \ll |Y|$, (2) $|X| \approx |Y|$, (3) Low channel bandwith.