Efficient Batched Oblivious PRF with Applications to Private Set Intersection

Vladimir Kolesnikov
Bell Labs
Murray Hill, New Jersey
kolesnikov@research.bell-labs.com

Ranjit Kumaresan
MIT
Cambridge, Massachusetts
vranjit@gmail.com

Mike Rosulek
Oregon State University
Corvallis, Oregon
rosulekm@eecs.oregonstate.edu

Ni Trieu
Oregon State University
Corvallis, Oregon
trieun@eecs.oregonstate.edu

ABSTRACT

We describe a lightweight protocol for oblivious evaluation of a pseudorandom function (OPRF) in the presence of semi-honest adversaries. In an OPRF protocol a receiver has an input r; the sender gets output s and the receiver gets output $F(s, r)$, where F is a pseudorandom function and s is a random seed. Our protocol uses a novel adaptation of 1-out-of-2 OT-extension protocols, and is particularly efficient when used to generate a large batch of OPRF instances. The cost to realize m OPRF instances is roughly the cost to realize $3.5m$ instances of standard 1-out-of-2 OTs (using state-of-the-art OT extension).

We explore in detail our protocol's application to semi-honest secure private set intersection (PSI). The fastest state-of-the-art PSI protocol (Pinkas et al., Usenix 2015) is based on efficient OT extension. We observe that our OPRF can be used to remove their PSI protocol's dependence on the bit-length of the parties' items. We implemented both PSI protocol variants and found ours to be 3.1–3.6× faster than Pinkas et al. for PSI of 128-bit strings and sufficiently large sets. Concretely, ours requires only 3.8 seconds to securely compute the intersection of 2^{20}-size sets, regardless of the bit length of the items. For very large sets, our protocol is only 4.3× slower than the *insecure* naïve hashing approach for PSI.

1. INTRODUCTION

This work involves OT, OPRF and PSI constructions. We start by reviewing the three primitives.

Oblivious Transfer.

Oblivious Transfer (OT) has been a central primitive in the area of secure computation. Indeed, the original protocols of Yao [30] and GMW [7, 8] both use OT in a critical manner. In fact, OT is both necessary and sufficient for secure computation [15]. Until early 2000's, the area of generic secure computation was often seen mainly as a feasibility exercise, and improving OT performance was not a priority research direction. This changed when Yao's Garbled Circuit (GC) was first implemented [20] and a surprisingly fast OT protocol (which we will call IKNP) was devised by Ishai et al. [12].

The IKNP OT extension protocol [12] is truly a gem; it allows 1-out-of-2 OT execution at the cost of computing and sending only a few hash values (but a security parameter of public key primitives evaluations were needed to bootstrap the system). IKNP was immediately noticed and since then universally used in implementations of the Yao and GMW protocols. It took a few years to realize that OT extension's use goes far beyond these fundamental applications. Many aspects of secure computation were strengthened and sped up by using OT extension. For example, Nielsen et al. [24] propose an approach to malicious two-party secure computation, which relates outputs and inputs of OTs in a larger construction. They critically rely on the low cost of batched OTs. Another example is the application of information-theoretic Gate Evaluation Secret Sharing (GESS) [16] to the computational setting [17]. The idea of [17] is to stem the high cost in secret sizes of the GESS scheme by evaluating the circuit by shallow slices, and using OT extension to efficiently "glue" them together. Particularly relevant for our work, efficient OTs were recognized by Pinkas et al. [28] as an effective building block for private set intersection, which we discuss in more detail later.

The IKNP OT extension, despite its wide and heavy use, received very few updates. In the semi-honest model it is still state-of-the-art. Robustness was added by Nielsen [23], and in the malicious setting it was improved only very recently [2, 14]. Improvement for short secret sizes, motivated by the GMW use case, was proposed by Kolesnikov and Kumaresan [18]. We use ideas from their protocol, and refer to it as the KK protocol. Under the hood, KK [18] noticed that one core aspect of IKNP data representation can be abstractly seen as a repetition error-correcting code, and their improvement stems from using a better code. As a result,

CCS'16, October 24-28, 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4139-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2976749.2978381

instead of 1-out of-2 OT, a 1-out of-n OT became possible at nearly the same cost, for n up to approximately 256.

Oblivious PRFs.

An oblivious pseudorandom function (OPRF) [6] is a protocol in which a sender learns (or chooses) a random PRF seed s while the receiver learns $F(s, r)$, the result of the PRF on a single input r chosen by the receiver. While the general definition of an OPRF allows the receiver to evaluate the PRF on several inputs, in this paper we consider only the case where the receiver can evaluate the PRF *on a single input.*

The central primitive of this work, an efficient OPRF protocol, can be viewed as a variant of Oblivious Transfer (OT) of random values. We build it by modifying the core of OT extension protocols [12, 18], and its internals are much closer to OT than to prior works on OPRF. Therefore, our presentation is OT-centric, with the results stated in OPRF terminology.

OT of random messages shares many properties with OPRF. In OT of random messages, the sender learns random m_0, m_1 while the receiver learns m_r for a choice bit $r \in \{0,1\}$. One can think of the function $F((m_0, m_1), r) = m_r$ as a pseudorandom function with input domain $\{0,1\}$. Similarly, one can interpret 1-out-of-n OT of random messages as an OPRF with input domain $\{1, \ldots, n\}$.

In this work, we propose a novel extension to the IKNP and KK protocols. At almost the same cost as 1-out-of-2 IKNP and KK OTs, we are able to achieve an 1-out-of-n OT of random messages for arbitrarily large n. This can be viewed as an OPRF with *unbounded* input domain $\{0,1\}^*$. That is, the receiver has an input $r \in \{0,1\}^*$ and learns the value $R(r)$, while the sender obtains the ability to evaluate $R(r')$ for any string r', where R is a pseudorandom function.

We call our main protocol **batched, related-key OPRF (BaRK-OPRF)** since it achieves a large number of OPRF instances, with keys that are related (in a way we describe later). This is a new primitive, which nevertheless suffices for the application to private set intersection that we consider.

Application to Private Set Intersection (PSI).

Private set intersection (PSI) refers to the setting where two parties each hold sets of items and wish to learn nothing more than the intersection of these sets. Today, PSI is a truly practical primitive, with extremely fast cryptographically secure implementations [27]. Incredibly, these implementations are only a relatively small factor slower than than the naïve and insecure method of exchanging hashed values. Among the problems of secure computation, PSI is probably the one most strongly motivated by practice. Indeed, already today companies such as Facebook routinely share and mine shared information [25, 31]. In 2012, (at least some of) this sharing was performed with insecure naive hashing. Today, companies are able and willing to tolerate a reasonable performance penalty, with the goal of achieving stronger security [31]. We believe that the ubiquity and the scale of private data sharing, and PSI in particular, will continue to grow as big data becomes bigger and privacy becomes a more recognized issue. We refer reader to [27, 28] for additional discussion and motivation of PSI.

In our work, we significantly improve state-of-the-art PSI protocol of [27] by replacing one of its components with BaRK-OPRF. This change results in a factor 2.3–3.6× performance improvement for PSI of moderate-length strings (64 or 128 bits) and reasonably large sets. We substantiate our algorithmic results with implementation and detailed evaluation. Our largest improvement is for the case of larger sets (2^{24} items each) of long strings (128 bits), which requires only one minute in our protocol but 214 seconds using [27].

1.1 Related work

Oblivious transfer.

Our BaRK-OPRF protocol can be seen as an OPRF protocol as well as a variant of oblivious transfer in the paradigm of IKNP [12]. As mentioned in Section 1, given its critical importance in secure computation, the IKNP OT extension has a surprisingly short list of follow up improvements, extensions and generalizations.

Most relevant prior work for us is the KK protocol [18], which views the IKNP OT from a new angle and presents a framework generalizing IKNP. More specifically, under the hood, players in the IKNP protocol encode Receiver's selection bit b as a repetition string of k copies of b. KK generalized this and allowed the use of an error-correcting code (ECC) with large distance as the selection bit encoding. For a code consisting of n codewords, this allowed to do 1-out of-n OT with consuming a single row of the OT extension matrix. In this work, we take the coding-theoretic perspective to the extreme. We observe that we never need to decode codewords, and by using (pseudo-)random codes we are able to achieve what amounts to a 1-out-of-poly OT by consuming a single row of the OT matrix, which for the same security guarantee is only about 3.5× longer than in the original IKNP protocol.

Our work is strictly in the semi-honest security model. Other work on OT extension extends the IKNP protocol to the malicious model [2, 14] and the PVC (publicly verifiable covert) model [19].

Oblivious PRF.

Oblivious pseudorandom functions were introduced by Freedman, Ishai, Pinkas, & Reingold [6]. In general, the most efficient prior protocols for OPRF require expensive public-key operations because they are based on algebraic PRFs. For example, an OPRF of [6] is based on the Naor-Reingold PRF [22] and therefore requires exponentiations. Furthermore, it requires a number of OTs proportional to the bit-length of the PRF input. The protocol of [3] constructs an OPRF from unique blind signature schemes. The protocol of [13] obliviously evaluates a variant of the Dodis-Yampolskiy PRF [4] and hence requires exponentiations (as well as other algebraic encryption components to facilitate the OPRF protocol).

Private set intersection.

Oblivious PRFs have many applications, but in this paper we explore in depth the application to private set intersection (PSI). We consider only the semi-honest security model. Our PSI protocol is most closely related to that of Pinkas et al. [27], which is itself an optimized variant of a previous protocol of [28]. We describe this protocol in great detail in Section 5.

We refer the reader to [28] for an overview of the many different protocol paradigms for PSI. As we have mentioned, the OT-based protocols have proven to be the fastest in practice. We do, however, point out that the OT-based protocols do not have the lowest *communication* cost. In settings where computation is not a factor, but communication is at a premium, the best protocols are those in the Diffie-Hellman paradigm introduced in [11]. In the semi-honest version of these protocols, each party sends only $2n$ group elements, where n is the number of items in each set. However, these protocols require a number of exponentiations proportional to the number of items, making their performance slow in practice. Concretely, [27] found Diffie-Hellman-based protocols to be over 200× slower than the OT-based ones.

While we closely follow the paradigm of [28], we abstract parts of their protocol in the language of oblivious PRFs (OPRF). The connection between OPRF and PSI was already pointed out in [6]. However, the most straightforward way of using OPRF to achieve PSI requires an OPRF protocol in which the receiver can evaluate the PRF on many inputs, whereas our OPRF allows only a single evaluation point for the receiver. OPRFs have been used for PSI elsewhere, generally in the malicious adversarial model [13, 10, 9].

Other applications of OPRF.

Just like a standard OPRF, our BaRK-OPRF variant immediately and efficiently implies the keyword search functionality of [6] (also called "string-select OT (SOT)" in [17]). Keyword search allows the receiver $\mathcal{R}$ to select the received secret via a string. In SOT the sender $\mathcal{S}$ has a mapping of keywords to secret values. $\mathcal{R}$ receives the secret corresponding to the keyword string it selected. In [17], SOT for k-bit selection strings is built by executing k 1-out of-2 OTs, and this technique is also essentially what is used in the PSI protocol of [28]. Using BaRK-OPRF, we can achieve keyword search by consuming only a single row of the OT extension matrix.

Oblivious PRFs can be used for *secure pattern matching* [10, 5], where one party holds a long text T and the other party holds a short pattern string p. The parties learn the location of all occurrences of p within T.

2. TECHNICAL OVERVIEW OF OUR RESULTS

We start with the OT-extension paradigm of Ishai, Kilian, Nissim & Petrank (IKNP) [12]. The goal of OT extension is to use a small number k of "base-OTs," plus only symmetric-key operations, to achieve $m \gg k$ "effective OTs." Here, k is chosen depending on the computational security parameter κ; in the following we show to what value k should be set. Below we describe an OT extension that achieves m 1-out-of-2 OTs of random strings, in the presence of semi-honest adversaries.

We follow the notation of [18] as it explicates the coding-theoretic framework for OT extension. Suppose the receiver has choice bits $r \in \{0,1\}^m$. He chooses two $m \times k$ matrices (m rows, k columns), T and U. Let $\boldsymbol{t}_j, \boldsymbol{u}_j \in \{0,1\}^k$ denote the j-th row of T and U, respectively. The matrices are chosen at random, so that:

$$\boldsymbol{t}_j \oplus \boldsymbol{u}_j = r_j \cdot 1^k \overset{\text{def}}{=} \begin{cases} 1^k & \text{if } r_j = 1 \\ 0^k & \text{if } r_j = 0 \end{cases}$$

The sender chooses a random string $s \in \{0,1\}^k$. The parties engage in k instances of 1-out-of-2 string-OT, *with their roles reversed*, to transfer to sender $\mathcal{S}$ the columns of either T or U, depending on the sender's bit s_i in the string s it chose. In the i-th OT, the receiver gives inputs $\boldsymbol{t}^i$ and $\boldsymbol{u}^i$, where these refer to the i-th *column* of T and U, respectively. The sender uses s_i as its choice bit and receives output $\boldsymbol{q}^i \in \{\boldsymbol{t}^i, \boldsymbol{u}^i\}$. Note that these are OTs of strings of length $m \gg k$ — the *length* of OT messages is easily extended. This can be done, e.g., by encrypting and sending the two m-bit long strings, and using OT on short strings to send the right decryption key.

Now let Q denote the matrix obtained by the sender, whose columns are $\boldsymbol{q}^i$. Let $\boldsymbol{q}_j$ denote the jth row. The key observation is that

$$\boldsymbol{q}_j = \boldsymbol{t}_j \oplus [r_j \cdot s] = \begin{cases} \boldsymbol{t}_j & \text{if } r_j = 0 \\ \boldsymbol{t}_j \oplus s & \text{if } r_j = 1 \end{cases} \tag{1}$$

Let H be a random oracle (RO). We have that the sender can compute two random strings $H(\boldsymbol{q}_j)$ and $H(\boldsymbol{q}_j \oplus s)$, of which the receiver can compute only one, via $H(\boldsymbol{t}_j)$. Note that $\boldsymbol{t}_j$ equals either $\boldsymbol{q}_j$ or $\boldsymbol{q}_j \oplus s$, depending on the receiver's choice bit r_j. Note that the receiver has no information about s, so intuitively he can learn only one of the two random strings $H(\boldsymbol{q}_j), H(\boldsymbol{q}_j \oplus s)$. Hence, each of the m rows of the matrix can be used to produce a single 1-out-of-2 OT.

As pointed out by [12], it is sufficient to assume that H is a correlation-robust hash function, a weaker assumption than RO. A special assumption is required because the same s is used for every resulting OT instance. See Section 3 for definition of correlation-robustness.

Coding interpretation.

In IKNP, the receiver prepares secret shares of T and U such that each row of $T \oplus U$ is either all zeros or all ones. Kolesnikov & Kumaresan [18] interpret this aspect of IKNP as a *repetition code* and suggest to use other codes instead.

Consider how we might use the IKNP OT extension protocol to realize 1-out-of-2^ℓ OT. Well, instead of a choice bit r_i for the receiver, r_i will now be an ℓ-bit string. Let C be a linear error correcting code of dimension ℓ and codeword length k. The receiver will prepare matrices T and U so that $\boldsymbol{t}_j \oplus \boldsymbol{u}_j = C(r_j)$.

Now, generalizing Equation 1 the sender receives

$$\boldsymbol{q}_j = \boldsymbol{t}_j \oplus [C(r_j) \cdot s] \tag{2}$$

where "$\cdot$" now denotes bitwise-AND of two strings of length k. (Note that when C is a repetition code, this is exactly Equation 1.)

For each value $r' \in \{0,1\}^\ell$, the sender associates the secret value $H(\boldsymbol{q}_j \oplus [C(r') \cdot s])$, which it can compute for all $r' \in \{0,1\}^\ell$. At the same time, the receiver can compute one of these values, namely, $H(\boldsymbol{t}_j)$. Rearranging Equation 2, we have:

$$H(\boldsymbol{t}_j) = H(\boldsymbol{q}_j \oplus [C(r_j) \cdot s])$$

Hence, the value that the receiver can learn is the secret value that the sender associates with the receiver's choice string $r' = r_j$.

At this point, OT of random strings is completed. For OT of chosen strings, the sender will use each $H(\boldsymbol{q}_i \oplus [C(r) \cdot s])$ as a key to encrypt the r'th OT message. The receiver will be able to decrypt only one of these encryptions, namely one corresponding to its choice string r_j.

To argue that the receiver learns *only one* string, suppose the receiver has choice bits r_j but tries to learn also the secret $H(\boldsymbol{q}_j \oplus [C(\tilde{r}) \cdot s])$ corresponding to a different choice $\tilde{r}$. We observe:

$$\begin{aligned} \boldsymbol{q}_j \oplus [C(\tilde{r}) \cdot s] &= \boldsymbol{t}_j \oplus [C(r_j) \cdot s] \oplus [C(\tilde{r}) \cdot s] \\ &= \boldsymbol{t}_j \oplus [(C(r_j) \oplus C(\tilde{r})) \cdot s] \end{aligned} \tag{3}$$

Importantly, everything in this expression is known to the receiver except for s. Now suppose the minimum distance of C is κ (the security parameter). Then $C(r_j) \oplus C(\tilde{r})$ has Hamming weight at least κ. Intuitively, the adversary would have to guess at least κ bits of the secret s in order to violate security. The protocol is secure in the RO model, and can also be proven under the weaker assumption of correlation robustness, following [12, 18].

Finally, we remark that the width k of the OT extension matrix is equal to the length of codewords in C. The parameter k determines the number of base OTs and the overall cost of the protocol.

Pseudorandom codes.

The main technical observation we make in this work is pointing out that the code C need not have many of the properties of error-correcting codes. In particular,

- We make no use of decoding, thus our code does not need to be efficiently decodable.
- We require only that for all possibilities r, r', the value $C(r) \oplus C(r')$ has Hamming weight at least equal to the computational security parameter κ. In fact, it is sufficient even if the Hamming distance guarantee is only probabilistic — i.e., it holds with overwhelming probability over choice of C (we discuss subtleties below).

For ease of exposition, imagine letting C be a random oracle with suitably long output. (Later we will show that C can be instantiated from a pseudorandom function in a straightforward way.) Intuitively, when C is sufficiently long, it should be hard to find a "near-collision." That is, it should be hard to find values r and r' such that $C(r) \oplus C(r')$ has low (less than a computational security parameter κ) Hamming weight. Later in Table 2 we compute the parameters more precisely, but for now we simply point out that a random function with output length $k = 4\kappa$ suffices to make near-collisions negligible in our applications.

We refer to such a function C (or family of functions, in our standard-model instantiation) as a **pseudorandom code (PRC)**, since its coding-theoretic properties — namely, minimum distance — hold in a cryptographic sense.

By relaxing the requirement on C from an error-correcting code to a pseudorandom code, we *remove the a-priori bound* on the size of the receiver's choice string! In essence, the receiver can use *any string* as its choice string; the sender can associate a secret value $H(\boldsymbol{q}_j \oplus [C(r') \cdot s])$ for any string r'. As discussed above, the receiver is only able to compute $H(\boldsymbol{t}_j) = H(\boldsymbol{q}_j \oplus [C(r) \cdot s])$ — the secret corresponding to its choice string r. The property of the PRC is that, with overwhelming probability, all other values of $\boldsymbol{q}_j \oplus [C(\tilde{r}) \cdot s]$ (that a polytime player may ever ask) differ from $\boldsymbol{t}_j$ in a way that would require the receiver to guess at least κ bits of s.

Interpretation as an oblivious PRF variant.

As discussed in Section 1, we can view the functionality achieved by this protocol as a kind of oblivious PRF. Intuitively, $r \mapsto H(\boldsymbol{q} \oplus [C(r) \cdot s])$ is a function that the sender can evaluate on any input, whose outputs are pseudorandom, and which the receiver can evaluate only on its chosen input r.

In Section 3 we give a formal definition of the functionality that we achieve. The main subtleties of the definition are:

1. the fact that the receiver learns slightly more than the output of this "PRF" — in particular, the receiver learns $\boldsymbol{t} = \boldsymbol{q} \oplus [C(r) \cdot s]$ rather than $H(\boldsymbol{t})$;
2. the fact that the protocol realizes many instances of this "PRF" but with related keys — s and C are shared among all instances.

We prove our construction secure assuming C is a pseudorandom code and that H satisfies a natural generalization of the "correlation robust" assumption from [12].

Summary & cost.

With our new variant of the IKNP protocol, we can obtain m OPRF instances efficiently, using only k base OTs plus symmetric-key operations. Compared to IKNP-paradigm OT extension for 1-out-of-2 OTs, the main differences in cost are:

- Cost associated with the increased width of the OT extension matrices. In our case, the matrix has width k rather than κ — concretely $3\kappa < k < 4\kappa$ in our applications. Note that the parameter k controls the number of base OTs required.[1]
- Computational costs associated with the pseudorandom code C. While in IKNP C is a repetition code, and in [18] C is a short Walsh-Hadamard code, in our protocol C is cryptographic. However, we are able to instantiate C using a PRF. In practice, we use AES as the PRF, and the associated hardware acceleration for AES in modern processors makes the cost of computing C minimal.

Application to private set intersection.

Private set intersection (PSI) refers to a computation in which Alice has a set A of items, Bob has a set B of items, and the two learn only $A \cap B$ and nothing more.

We show how BaRK-OPRF can be used to significantly reduce the cost of semi-honest-secure PSI. The current fastest protocol for the task is that of Pinkas et al. [27]. The protocol relies heavily on efficient OT extension (for standard 1-out-of-2 OTs).

Looking closely at the PSI protocol of [27], we see that they use a number of OTs that is proportional to $N\ell$, where N is the number of items in the parties' sets and ℓ is the length (in bits) of those items. We can replace their use of 1-out-of-2 OTs with a suitable use of BaRK-OPRF and

[1] In our instantiation, we actually use IKNP to extend κ base OTs to k OTs, and then use those k OTs as base OTs for BaRK-OPRF instances. Hence, the number of public-key OT operations is unchanged. Still, the total communication cost remains proportional to km in our protocol rather than κm.

remove the dependence on ℓ. Our protocol uses a number of BaRK-OPRF instances that is proportional only to N.

We implemented our BaRK-OPRF-based PSI protocol and compared its performance to that of [27]. For PSI on strings of length $\ell \in \{64, 128\}$ and sufficiently large sets, our protocol is 2.3–3.6 times faster. This is a significant achievement in the already very polished PSI state of the art!

3. TECHNICAL PRELIMINARIES

We write $\|x\|_{\mathsf{H}}$ to denote the hamming weight of a binary string x. Our computational security parameter is κ and statistical security parameter is σ.

3.1 Correlation Robustness

The OT extension protocol of IKNP [12] is proven secure under a so-called *correlation robustness* assumption on the underlying hash function. Our protocol makes use of the following generalization of this notion:

DEFINITION 1. *Let H be a hash function with input length n. Then H is **d-Hamming correlation robust** if, for any strings $z_1, \dots, z_m \in \{0,1\}^*$, $a_1, \dots, a_m, b_1, \dots, b_m \in \{0,1\}^n$ with $\|b_i\|_H \geq d$, the following distribution, induced by random sampling of $s \leftarrow \{0,1\}^n$, is pseudorandom:*

$$H(z_1 \| a_1 \oplus [b_1 \cdot s]), \dots, H(z_m \| a_m \oplus [b_m \cdot s])$$

As in the overview, "$\cdot$" denotes bitwise-AND.

The definition generalizes previous ones in the following way:

- If $d = n$, then the only legal choice of b_i is 1^n, and $H(z_i \| a_i \oplus [b_i \cdot s])$ simplifies to $H(z_i \| a_i \oplus s)$. Restricting the definition in this way, and taking $z_i = i$ corresponds to the IKNP notion of correlation robustness.
- If the b_i values are required to be elements of a linear error correcting code $\mathcal{C}$, then the resulting definition is one under which the construction of [18] is secure (for simplicity they prove security in the random oracle model).

3.2 Pseudorandom Codes

We now formalize the notion of a pseudorandom code, motivated in Section 2.

DEFINITION 2. *Let $\mathcal{C}$ be a family of functions. We say that $\mathcal{C}$ is a (d, ϵ) **pseudorandom code (PRC)** if for all strings $x \neq x'$,*

$$\Pr_{C \leftarrow \mathcal{C}}\left[\|C(x) \oplus C(x')\|_H < d\right] \leq 2^{-\epsilon}$$

That is, a (d,ϵ)-PRC guarantees that the hamming distance of two codewords is less or equal to d with probability at most $2^{-\epsilon}$.

The reader may find it convenient to think of C as a random oracle. However, it suffices for C to be a pseudorandom function instantiated with random seed:

LEMMA 1. *Suppose $F : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ is a pseudorandom function. Define $\mathcal{C} = \{F(s, \cdot) \mid s \in \{0,1\}^\kappa\}$. Then $\mathcal{C}$ is a (d,ϵ)-pseudorandom-code where:*

$$2^{-\epsilon} = 2^{-n} \sum_{i=0}^{d-1} \binom{n}{i} + \nu(\kappa).$$

and ν is a negligible function.

PROOF. Consider the following game. An adversary has strings x and x' hard-coded. It queries its oracle $\mathcal{O}$ on x and x' and outputs 1 if $\mathcal{O}(x)$ and $\mathcal{O}(x')$ are within Hamming distance d.

When $\mathcal{O}$ is instantiated as a random function, a simple counting argument shows that the adversary outputs 1 with probability $2^{-n} \sum_{i=0}^{d-1} \binom{n}{i}$.

When $\mathcal{O}$ is instantiated as a PRF F with random seed, the probability must be within $\nu(\kappa)$ of the above probability, where ν is negligible. The adversary's output probability in this instantiation is exactly the probability specified in the PRC security definition, so the lemma follows.

Note that in our typical usage of PRCs, the choice of C (in this case, the seed to the PRF) is a public value. But in both the security definition for PRC and in this analysis, the values x and x' are fixed before the PRF key is chosen. Whether or not $F(s,x)$ and $F(s,x')$ are within Hamming distance d is not affected by making the PRF seed public. □

3.3 Our Oblivious PRF Variant

As outlined in Section 2, our main construction is a variant of OT-extension which associates a pseudorandom output $R(x)$ for every possible input $r \in \{0,1\}^*$. The sender can compute $R(r)$ for any r, while the receiver learns $R(x)$ for only a single value r. This functionality is reminiscent of an **oblivious PRF (OPRF)** [6]. In this section we describe how our construction can be interpreted as a variant OPRF functionality.

In an OPRF functionality for a PRF F, the receiver provides an input[2] r; the functionality chooses a random seed s, gives s to the sender and $F(s,r)$ to the receiver.

In our protocol, the sender knows $\boldsymbol{q}_j$ and s. We can consider these values as keys to a PRF:

$$F((\boldsymbol{q}_j, s), r) = H(j \| \boldsymbol{q}_j \oplus [C(r) \cdot s])$$

Intuitively, the sender can evaluate this PRF at any point, while the receiver can evaluate it on only one. However, we point out some subtleties:

- In our protocol, the receiver learns $\boldsymbol{t}_j = \boldsymbol{q}_j \oplus [C(r^*) \cdot s]$ for his chosen input r^*, which is more information than the "PRF output" $H(j \| \boldsymbol{t}_j)$. However, even knowing $\boldsymbol{t}_j$, the other outputs of the "PRF" still look random. This common feature of an OPRF protocol leaking slightly more than the PRF output is called *relaxed OPRF* in [6].
- In our protocol, we realize many "OPRF" instances *with related keys.* In particular, all instances have the same component s (and C).

We encapsulate these properties in the following definitions.

3.3.1 Our PRF variant

We refer to F as a **relaxed PRF** if there is another function $\widetilde{F}$, such that $F(k,r)$ can be efficiently computed given just $\widetilde{F}(k,r)$. We then define the relevant notion of security with respect to an adversary who can query the relaxed function $\widetilde{F}$ rather than just F.

[2] More general OPRF variants allow the receiver to learn the PRF output on many inputs — here it suffices to limit the receiver to one input.

Definition 3. *Let F be a relaxed PRF with output length v, for which we can write the seed as a pair (k^*, k). Then F has* m**-related-key-PRF** (m**-RK-PRF**) **security** *if the advantage of any PPT adversary in the following game is negligible:*

1. *The adversary chooses strings $x_1, \ldots, x_n$ and m pairs $(j_1, y_1), \ldots, (j_m, y_m)$, where $y_i \neq x_{j_i}$.*
2. *Challenger chooses random values appropriate for PRF seeds $k^*, k_1, \ldots, k_n$ and tosses a coin $b \leftarrow \{0,1\}$.*
   (a) *If $b = 0$, the challenger outputs $\{\widetilde{F}((k^*, k_j), x_j)\}_j$ and $\{F((k^*, k_{j_i}), y_i)\}_i$.*
   (b) *If $b = 1$ the challenger chooses $z_1, \ldots, z_m \leftarrow \{0,1\}^v$ and outputs $\{\widetilde{F}((k^*, k_j), x_j)\}_j$ and $\{z_i\}_i$,*
3. *The adversary outputs a bit b'. The advantage of the adversary is* $\Pr[b = b'] - 1/2$. ♢

Intuitively, the PRF is instantiated with n related keys (sharing the same k^* value). The adversary learns the *relaxed* output of the PRF on one chosen input for each key. Then any m additional PRF outputs (corresponding to any seed) are indistinguishable from random by the adversary.

Lemma 2. *Let $\mathcal{C}$ be a $(d, \epsilon + \log_2 m)$-PRC, where $1/2^\epsilon$ is a negligible function, Let H be a d-Hamming correlation robust hash function. Define the following relaxed PRF, for $C \in \mathcal{C}$:*

$$F\Big(((C, s), (\boldsymbol{q}, j)), r\Big) = H(j \| \boldsymbol{q} \oplus [C(r) \cdot s])$$
$$\widetilde{F}\Big(((C, s), (\boldsymbol{q}, j)), r\Big) = (j, C, \boldsymbol{q} \oplus [C(r) \cdot s])$$

Then F has m-RK-PRF security.

Proof. In the m-RK-PRF game with this PRF, we can rewrite the adversary's view as in Section 2 as:

$$(C, \{\boldsymbol{t}_j\}_j, \{H(j_i \| \boldsymbol{t}_{j_i} \oplus [(C(x_{j_i}) \oplus C(y_i)) \cdot s])\}_i)$$

There are m terms of the form $C(x_{j_i}) \oplus C(y_i)$ for $x_{j_i} \neq y_i$. Each of these terms has Hamming weight at least d with probability at least $1 - 2^{-\epsilon - \log_2 m}$ over the choice of C. By a union bound, *all* m terms have Hamming weight at least d with probability $1 - 2^{-\epsilon}$. Conditioning on this (overwhelmingly likely) event, we can apply the d-Hamming correlation robust property of H to see that the H-outputs are indistinguishable from random. □

3.3.2 *Our* BaRK-OPRF *functionality*

In Figure 1 we formally describe the variant OPRF functionality we achieve. It generates m instances of the PRF with related keys, and allows the receiver to learn the (relaxed) output on one input per key.

4. MAIN CONSTRUCTION

We present our main construction, which is a semi-honest secure protocol for the functionality in Figure 1, instantiated with the relaxed PRF defined in Lemma 2.

The functionality is parameterized by a relaxed PRF F, a number m of instances, and two parties: a **sender** and **receiver**.

On input $(r_1, \ldots, r_m)$ from the receiver,

- Choose random components for seeds to the PRF: $k^*, k_1, \ldots, k_m$ and give these to the sender.
- Give $\widetilde{F}((k^*, k_1), r_1), \ldots, \widetilde{F}((k^*, k_m), r_m)$ to the receiver.

Figure 1: Batched, related-key OPRF (BaRK-OPRF) ideal functionality.

4.1 Notation

We use the notation OT_m^k to denote k instances of 1-out-of-2 string-OT where the strings are m bits long. Let $\mathcal{S}$ denote the sender, and let $\mathcal{R}$ denote the receiver. In OT_m^k, the sender's input is $\{(x_{j,0}, x_{j,1})\}_{j \in [k]}$, i.e., m pairs of strings, each of length m, and the receiver holds input $\{r_j\}_{j \in [k]}$, where each r_j is a choice bit. Note that if $\mathcal{S}$ provides input $\{(x_{j,0}, x_{j,1})\}_{j \in [k]}$ to OT_m^k, and if $\mathcal{R}$ provides input $\{r_j\}_{j \in [k]}$ to OT_m^k, then $\mathcal{R}$ receives back $\{x_{j,r_j}\}_{j \in [k]}$, while $\mathcal{S}$ receives nothing.

Following the convention in IKNP, we denote vectors in bold, and matrices in capitals. For a matrix A, we let $\mathbf{a}_j$ denote the j-th row of A, and $\mathbf{a}^i$ denote the i-th column of A. If $\mathbf{a} = a_1 \| \cdots \| a_p$ and $\mathbf{b} = b_1 \| \cdots \| b_p$ are two vectors, then we define $\oplus$ and $\cdot$ operations as follows. We use the notation $\mathbf{a} \oplus \mathbf{b}$ to denote the vector $(a_1 \oplus b_1) \| \cdots \| (a_p \oplus b_p)$. Similarly, the notation $\mathbf{a} \cdot \mathbf{b}$ denotes the vector $(a_1 \cdot b_1) \| \cdots \| (a_p \cdot b_p)$. Finally, suppose $c \in \{0,1\}$, then $c \cdot \mathbf{a}$ denotes the vector $(c \cdot a_1) \| \cdots \| (c \cdot a_p)$.

We note that to simplify notation via indexing, in the following we will refer to the OT matrices as T_0 and T_1, rather than as T and U, as we did when presenting high-level overview of our work.

4.2 The BaRK-OPRF construction

Our BaRK-OPRF protocol is presented in Figure 2. It closely follows the high-level overview. Recall that we are considering a PRF whose seed is of the form $((C, \boldsymbol{s}), (j, \boldsymbol{q}_j))$ and whose relaxed output is of the form $\boldsymbol{t}_{0,j} = \boldsymbol{q}_j \oplus (C(r_j) \cdot \boldsymbol{s})$.

Theorem 3. *The* BaRK-OPRF *protocol in Figure 2 securely realizes the functionality of Figure 1, instantiated with the relaxed PRF defined in Lemma 2, in the presence of semi-honest adversaries, where κ is the computational security parameter.*

Proof. When using the abstraction of our OPRF functionality, the proof is elementary.

Simulating $\mathcal{S}$. The simulator receives output from the OPRF ideal functionality consisting of related PRF seeds: a common $(C, \boldsymbol{s})$ and a $\boldsymbol{q}_j$ for each $j \in [m]$. Let Q be a matrix whose rows are the $\boldsymbol{q}_j$. Let $\boldsymbol{q}^i$ denote the ith column of Q.

The simulator simulates an execution of the protocol in which $\mathcal{S}$ chooses C in step 0, chooses $\boldsymbol{s}$ in step 1, and receives output $\{\boldsymbol{q}^i\}_{i \in [k]}$ as OT output in step 3.

Simulating $\mathcal{R}$. The simulator has input $(r_1, \ldots, r_m)$ and receives output from the OPRF ideal functionality consisting of a relaxed PRF output $(j, C, \boldsymbol{t}_j)$ for each $j \in [m]$.

INPUT OF $\mathcal{R}$: m selection strings $\boldsymbol{r} = (r_1, \ldots, r_m)$, $r_i \in \{0,1\}^*$.

PARAMETERS:

- A (κ, ϵ)-PRC family $\mathcal{C}$ with output length $k = k(\kappa)$.
- A κ-Hamming correlation-robust $H : [m] \times \{0,1\}^k \to \{0,1\}^v$.
- An ideal OT^k_m primitive.

PROTOCOL:

0. $\mathcal{S}$ chooses a random $C \leftarrow \mathcal{C}$ and sends it to $\mathcal{R}$.
1. $\mathcal{S}$ chooses $\boldsymbol{s} \leftarrow \{0,1\}^k$ at random. Let s_i denote the i-th bit of $\boldsymbol{s}$.
2. $\mathcal{R}$ forms $m \times k$ matrices T_0, T_1 in the following way:
   - For $j \in [m]$, choose $\boldsymbol{t}_{0,j} \leftarrow \{0,1\}^k$ and set $\boldsymbol{t}_{1,j} = C(r_j) \oplus \boldsymbol{t}_{0,j}$.

   Let $\boldsymbol{t}^i_0, \boldsymbol{t}^i_1$ denote the i-th column of matrices T_0, T_1 respectively.
3. $\mathcal{S}$ and $\mathcal{R}$ interact with OT^k_m in the following way:
   - $\mathcal{S}$ acts as *receiver* with input $\{s_i\}_{i \in [k]}$.
   - $\mathcal{R}$ acts as *sender* with input $\{\boldsymbol{t}^i_0, \boldsymbol{t}^i_1\}_{i \in [k]}$.
   - $\mathcal{S}$ receives output $\{\boldsymbol{q}^i\}_{i \in [k]}$.

   $\mathcal{S}$ forms $m \times k$ matrix Q such that the i-th column of Q is the vector $\boldsymbol{q}^i$. (Note $\boldsymbol{q}^i = \boldsymbol{t}^i_{s_i}$.) Let $\boldsymbol{q}_j$ denote the j-th row of Q. Note, $\boldsymbol{q}_j = ((\boldsymbol{t}_{0,j} \oplus \boldsymbol{t}_{1,j}) \cdot \boldsymbol{s}) \oplus \boldsymbol{t}_{0,j}$. Simplifying, $\boldsymbol{q}_j = \boldsymbol{t}_{0,j} \oplus (C(r_j) \cdot \boldsymbol{s})$.
4. For $j \in [m]$, $\mathcal{S}$ outputs the PRF seed $((C, \boldsymbol{s}), (j, \boldsymbol{q}_j))$.
5. For $j \in [m]$, $\mathcal{R}$ outputs relaxed PRF output $(C, j, \boldsymbol{t}_{0,j})$.

Figure 2: The BaRK-OPRF protocol

The simulator simulates an execution of the protocol in which $\mathcal{R}$ receives C in step 0 and samples $\boldsymbol{t}_{0,j} = \boldsymbol{t}_j$ in step 2.

In both cases it is straightforward to check that the simulation is perfect. □

5. IMPROVING PRIVATE SET INTERSECTION

The main application of BaRK-OPRF is to improve the performance of semi-honest-secure **private set intersection (PSI).** Pinkas et al. [28] give a thorough summary of many different paradigms for PSI in this model.

For our purposes, we summarize only the most efficient PSI protocol, which is the OT-based paradigm of [28] including the optimizations suggested in follow up work [27]. Hereafter we refer to their protocol as the "PSSZ" protocol.

5.1 The OPRF Implicit in PSSZ

The main building block of PSSZ, private equality test, can be viewed as a relaxed OPRF based on random OTs (i.e., oblivious transfers of random messages), which can be obtained efficiently from OT extension. The protocol is as follows, where Bob has input r, with $\ell = |r|$.

- The parties perform ℓ 1-out-of-2 OTs of random messages, with Alice as receiver. Bob acts as receiver and uses the bits of r as his choice bits. In the ith OT, Alice learns random strings $m_{i,0}$ and $m_{i,1}$, while Bob learns $m_{i,r[i]}$.
- Define the mapping $F(x) = H\left(\bigoplus_i m_{i,x[i]}\right)$, where H is a random oracle. One can then view F as a PRF whose keys are the $m_{i,b}$ values (known to Alice). Bob learns the output of F on r only. More precisely, he learns *relaxed* output $\{m_{i,r[i]}\}_i$, for which all other outputs of F are pseudorandom.

In this description, we have treated r as a string of bits, and therefore use 1-out-of-2 (random) OTs. However, when using the OT extension protocol of [18], the cost of a 1-out-of-2 random OT is essentially the same as a 1-out-of-256 random OT. Hence, PSSZ interpret r as strings of characters over $\{0,1\}^8$. The protocol uses one instance of 1-out-of-256 ROT for each *byte* (not bit) of r.

Regardless of whether one uses 1-out-of-2 or 1-out-of-256 OT, this OPRF protocol has cost that scales with length of the input r, whereas ours has cost independent of the input length. Our main improvement to PSSZ consists of replacing their OPRF with ours. The rest of the protocol is largely unchanged.

5.2 PSI from OPRF

We now describe how the PSSZ paradigm achieves PSI using an OPRF. This part of the overall PSI protocol is nearly identical between our implementation and that of [27] (we include an additional small optimization). For concreteness, we describe the parameters used in PSSZ when the parties have roughly the same number n of items.

The protocol relies on **Cuckoo hashing** [26] with 3 hash functions, which we briefly review now. To assign n items into b bins using Cuckoo hashing, first choose random functions $h_1, h_2, h_3 : \{0,1\}^* \to [b]$ and initialize empty bins $\mathcal{B}[1, \ldots, b]$. To hash an item x, first check to see whether any of the bins $\mathcal{B}[h_1(x)]$, $\mathcal{B}[h_2(x)]$, $\mathcal{B}[h_3(x)]$ are empty. If so, then place x in one of the empty bins and terminate. Otherwise, choose a random $i \in \{1,2,3\}$, evict the item currently in $\mathcal{B}[h_i(x)]$, replacing it with x, and then recursively try to insert the evicted item. If this process does not terminate after a certain number of iterations, then the final evicted element is placed in a special bin called the *stash*.

PSSZ use Cuckoo hashing for PSI in the following way. First, the parties choose 3 random hash functions h_1, h_2, h_3 suitable for 3-way Cuckoo hashing. Suppose Alice has a set X of inputs and Bob has a set Y, where $|X| = |Y| = n$. Bob maps his items into $1.2n$ bins using Cuckoo hashing and a stash of size s. At this point Bob has at most one item per bin and at most s items in his stash — he pads his input with dummy items so that each bin contains exactly 1 item and the stash contains exactly s items.

The parties then run $1.2n+s$ instances of an OPRF, where Bob plays the role of receiver and uses each of his $1.2n + s$ items as OPRF input. Let $F(k_i, \cdot)$ denote the PRF evaluated in the ith OPRF instance. If Bob has mapped item y to bin i via cuckoo hashing, then Bob learns $F(k_i, y)$; if Bob has mapped y to position j in the stash, then Bob learns $F(k_{1.2n+j}, y)$.

On the other hand, Alice can compute $F(k_i, \cdot)$ for any i. So she computes sets of candidate PRF outputs:

$$H = \{F(k_{h_i(x)}, x) \mid x \in X \text{ and } i \in \{1,2,3\}\}$$
$$S = \{F(k_{1.2n+j}, x) \mid x \in X \text{ and } j \in \{1, \ldots, s\}\}$$

She randomly permutes elements of H and elements of S and sends them to Bob. Bob can identify the intersection of X and Y as follows. If Bob has an item y mapped to the stash, he checks whether the associated OPRF output is present in S. If Bob has an item y *not* mapped to the stash, he checks whether its associated OPRF output is in H.

Intuitively, the protocol is secure against a semi-honest Bob by the PRF property. For an item $x \in X \setminus Y$, the corresponding PRF outputs $F(k_i, y)$ are pseudorandom. It is easy to see that security holds even if Bob learns *relaxed* PRF outputs and the PRF achieves RK-PRF security, Definition 3 (i.e., Alice's PRF outputs are pseudorandom to an adversary who learns *relaxed* PRF outputs). Similarly, if the PRF outputs are pseudorandom even under related keys, then it is safe for the OPRF protocol to instantiate the PRF instances with related keys.

The protocol is correct as long as the PRF does not introduce any further collisions (i.e., $F(k_i, x) = F(k_{i'}, x')$ for $x \neq x'$). Below we discuss the parameters required to prevent such collisions.

An Optimization.

In the protocol summary above, Bob must search for each of his OPRF outputs, either in the set H or the set S. Furthermore, $|H| = 3n$ and $|S| = sn$. Even when using a reasonable data structure for these comparisons, they have a non-trivial effect on the protocol's running time. We now describe an optimization that reduces this cost (by approximately 10% in our implementation). The full protocol is described in Figure 3.

Our modification works as follows. First, Bob keeps track of a hash index $z \in \{1,2,3\}$ for each item $y \in Y$ that is not mapped to the stash. For example, if Bob's Cuckoo hashing maps y to bin $\#h_2(y)$, then Bob associates $z = 2$ with y. If for example y is mapped to bin by two hash functions $\#h_1(y) = \#h_2(y)$ then Bob may choose either $z = 1$ or $z = 2$ arbitrarily.

Then in the first $1.2n$ OPRF instances, Bob uses input $y\|z$. For the OPRF instances associated with the stash, he does not need to append the index z. Summarizing, if Bob has mapped item to position j in the stash, then Bob learns $F(k_{1.2n+j}, y)$. If he has not mapped y to the stash, then he learns $F(k_{h_z(x)}, y\|z)$ for *exactly one* z.

Then Alice computes the following sets:

$$H_i = \{F(k_{h_i(x)}, x\|i) \mid x \in X\}, \text{ for } i \in \{1,2,3\}$$
$$S_j = \{F(k_{1.2n+j}, x) \mid x \in X\}, \text{ for } j \in \{1, \ldots, s\}$$

She randomly permutes the contents of each H_i and each S_j and sends them to Bob. For each item y of Bob, if y is not mapped to the stash then Bob can whether $F(k_{h_{z(y)}}, y\|z) \in H_z$, for the associated hash-index z. If his Cuckoo hashing maps item y to position j in the stash, he can check whether $F(k_{1.2n+j}, y) \in S_j$.

The reason for appending the hash-index z to the PRF input is as follows. Suppose $h_1(x) = h_2(x) = i$, which is indeed can happen with noticeable probability, since the output range of h_1, h_2 is small ($[1.2n]$). Without appending z, both H_1 and H_2 would contain the identical value $F(k_i, x)$. This would leak the fact that a collision $h_1(x) = h_2(x)$ occurred. Such an event is input-dependent so cannot be simulated.[3]

With our optimization: (1) All of the calls to the PRF made by Alice (to compute the H_i's and S_j's) invoke the PRF on *distinct* key-input pairs. This ensures that the contents of these sets can be easily simulated; (2) Bob searches for each of his PRF outputs within only one set (either an H_i or an S_j) of n items. Contrast this with the approach described previously, where Bob must find each OPRF output in either a set of size $3n$ or sn (depending on whether the item is in the stash or not).

Recall that the protocol is correct as long as there are no spurious collisions among PRF outputs. Since there are at most n^2 opportunities for a spurious collision (Bob searches each time for a PRF output in a set of n items), we can limit the overall probability of a spurious collision to $2^{-\sigma}$ by using PRF outputs of length $\sigma + \log_2(n^2)$.

Parameters: Alice has input X; Bob has input Y, with $|X| = |Y| = n$. s is an upper bound on the stash size for Cuckoo hashing.

1. Bob specifies random hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [1.2n]$ and tells them to Alice.

2. Bob assigns his items Y into $1.2n$ bins using Cuckoo hashing. Let Bob keep track of $z(y)$ for each y so that: if $z(y) = \perp$ then y is in the stash; otherwise y is in bin $h_{z(y)}(y)$. Arrange the items in the stash in an arbitrary order.

   Bob selects OPRF inputs as follows: for $i \in [1.2n]$, if bin $\#i$ is empty, then set r_i to a dummy value; otherwise if y is in bin $\#i$ then set $r_i = y\|z(y)$. For $i \in [s]$, if position i in the stash is y, then set $r_i = y$; otherwise set r_i to a dummy value.

3. The parties invoke $1.2n + s$ OPRF instances, with Bob the receiver with inputs $(r_1, \ldots, r_{1.2n+s})$. Alice receives $(k_1, \ldots, k_{1.2n+s})$ and Bob receives $F(k_i, r_i)$ for all i.

4. Alice computes:

   $$H_i = \{F(k_{h_i(x)}, x\|i) \mid x \in X\}, \text{ for } i \in \{1,2,3\}$$
   $$S_j = \{F(k_{1.2n+j}, x) \mid x \in X\}, \text{ for } j \in \{1, \ldots, s\}$$

   and sends a permutation of each set to Bob.

5. Bob initializes an empty set $\mathcal{O}$ and does the following for $y \in Y$: If $z(y) = \perp$ and y is at position j in the stash and $F(k_{1.2n+j}, y) \in S_j$, then Bob adds y to $\mathcal{O}$. If $z(y) \neq \perp$ and $F(k_{h_{z(y)}(y)}, y\|z(y)) \in H_{z(y)}$ then Bob adds y to $\mathcal{O}$.

6. Bob sends $\mathcal{O}$ to Alice and both parties output $\mathcal{O}$.

Figure 3: Our optimization to the PSSZ PSI protocol, written in terms of an OPRF functionality.

[3] The protocol and implementation of PSSZ do not account for such collisions among the Cuckoo hash functions. Duplicate values will appear in H in such an event.

5.3 Comparing OPRF Subprotocols

When comparing our protocol to that of PSSZ, the major difference is the choice of OPRF subprotocols. Later in Section 6 we give an empirical comparison of the protocols. For now, we derive an analytical comparison of the costs of the two OPRF subprotocols, to give a better sense of our improvement.

We focus on the *communication cost* associated with the OT primitives. Communication cost is an objective metric, and it often reflects the bottleneck in practice (especially in these protocols where essentially all of the cryptographic computations are precomputed). Although the computation costs of our protocols are different (e.g., ours requires computing the pseudorandom code, which is a cryptographic operation), communication cost is nonetheless a good proxy for computation costs in OT extension protocols. The data that is communicated in these protocols is a large matrix that must be transposed, and this transposition is the primary contributor to the computational cost.

The main benefit of our protocol is that its cost *does not scale with the size of the items being compared.* Each instance of OPRF consumes just one row of the OT extension matrix. The width of this OT extension matrix is exactly the length of the pseudorandom code (PRC). In Section 6.1 we describe how to compute an appropriate length of PRC. For the range of parameters we consider, this parameter is 424–448 bits. Hence the OT-cost of one instance of our OPRF protocol is 424–448 bits. The specific numbers are in Table 1.

The PSSZ OPRF protocol uses several instances of 1-out-of-256 ROT. With security parameter 128, the cost of such a random OT is 256 bits using the OT extension of [18].

The main optimization of [27] allows for the OPRF subprotocols to be performed on items of length $\ell^* = \ell - \log n$ (n is the number of items in the overall PSI protocol) rather than length ℓ. Let ℓ^* denote this *effective item length.* Then $\ell^*/8$ instances of 1-out-of-256 ROT are needed for one OPRF instance. The total OT-cost of their OPRF protocol is therefore $256\ell^*/8 = 32\ell^*$ bits.

Hence, we see that our protocol has lower communication cost whenever $\ell^* > 448/32 = 14$. Among the different parameter settings reported in [27], the only configuration with $\ell^* < 14$ is for PSI of $n \geq 2^{20}$ items of length 32 bits. For all other configurations, our PSI protocol has lower communication cost, with the savings increasing as the items become longer. See Table 1.

Remark on pre-hashing long PSI inputs.

Our improvements to PSI are most significant for PSI of long items. Yet, if the parties have items which are *very* long strings (say, thousands of bits), they can agree on a random hash function, hash their items locally, and perform PSI on the shorter hashes instead. The reader may rightfully wonder whether this idea make our improvements irrelevant!

For this approach (hash-then-PSI) to work, we must ensure that the hashing introduces no collisions among the parties' items. If the parties have n items each, and we wish to limit the probability of a collision to $2^{-\sigma}$, then we must choose a hash function whose length is $\sigma + 2\log n$. When using the optimizations of [27], the *effective* item length can be reduced from $\sigma + 2\log n$ to $\sigma + \log n$ bits.

We see that pre-hashing the items cannot reduce their *effective* length below σ bits, where σ is a statistical security parameter. Standard practice suggests $\sigma \geq 40$, and yet our protocol outperforms [27] whenever the *effective* item length is at least 14 bits. Hence hash-then-PSI does not allow one to bypass our improvement to [27].

n	ℓ	ℓ^*	OT cost PSSZ	 our BaRK-OPRF	ratio
2^8	32	24	768	**424**	0.54
2^8	64	56	1792	**424**	0.24
2^8	128	120	3840	**424**	0.11
2^{12}	32	20	640	**432**	0.68
2^{12}	64	52	1664	**432**	0.26
2^{12}	128	116	3712	**432**	0.12
2^{16}	32	16	576	**440**	0.76
2^{16}	64	48	1536	**440**	0.29
2^{16}	128	112	3584	**440**	0.12
2^{20}	32	12	**384**	448	1.17
2^{20}	64	44	1408	**448**	0.32
2^{20}	128	108	3456	**448**	0.13
2^{24}	32	8	**256**	448	1.75
2^{24}	64	40	1280	**448**	0.35
2^{24}	128	104	3328	**448**	0.13

Table 1: Comparing the OT-cost of PSSZ-paradigm OPRF subprotocol and ours, for various parameters. The entries in the table refer to the contribution (in bits) to the size of the OT-extension matrices. ℓ is the item length (in bits), n is the total number of items in the parties' sets, and ℓ^* is the *effective* item length when using the optimizations of [27].

On a similar note, in our experimental results we report performance of the protocols only for PSI inputs up to 128 bits long. For statistical security parameter $\sigma = 40$, as long as the parties have at most 2^{34} (17 billion) items, they can use hash-then-PSI with a 128-bit hash.

6. IMPLEMENTATION & PERFORMANCE

We implemented our PSI protocol and report on its performance in comparison with the state-of-the-art PSI protocol of [27]. Our complete implementation is available on GitHub: https://github.com/osu-crypto/BaRK-OPRF.

In our implementation we used parameter settings consistent with PSSZ or stricter, and ran their and our code on our system so as to obtain meaningful comparisons. As do PSSZ, we use matrix transposition code from [1] and several other optimizations.

6.1 Choosing Suitable Parameters

In this section we discuss concrete parameters used in our implementation. We use a computational security parameter of $\kappa = 128$ and a statistical security parameter of $\sigma = 40$. The other parameters are:

s: the maximum size of the stash for Cuckoo hashing, when hashing n items into $1.2n$ using 3 hash functions.

k: length of the pseudorandom code (and hence the width of the OT extension matrix) in the BaRK-OPRF protocol.

v: output length of the PRF realized by the BaRK-OPRF protocol.

n	s	k	v
2^8	12	424	56
2^{12}	6	432	64
2^{16}	4	440	72
2^{20}	3	448	80
2^{24}	2	448	88

Table 2: Parameters used in our implementation. n is the size of the parties' input sets; s is the maximum stash size for Cuckoo hashing; k is the width of the pseudorandom code (in bits); v is the length of OPRF output (in bits).

A summary of our concrete parameter choices is given in Table 2. Below we describe how these parameters were derived.

Hashing parameters.

Bob uses Cuckoo hashing with 3 hash functions to assign his n items into $1.2n$ bins (and a stash). For the appropriate choice of the stash size s, we use the numbers given in [27], which limit the probability of hashing failure to 2^{-40}.

Size of pseudorandom code.

Our BaRK-OPRF protocol requires a pseudorandom code achieving minimum distance $\kappa = 128$. In our protocol, Alice evaluates the PRF on $(3+s)n$ values. In order to argue that these values can be collectively pseudorandom, so we require the underlying PRF to have m-RK-PRF security (Definition 3) for $m = (3+s)n$.

From Lemma 2, this means we must choose a pseudorandom code with parameters $(d = \kappa, \epsilon = \sigma + \log m)$. Using Lemma 1, we calculate the minimum length of such a pseudorandom code; the results are column k in Table 2. We round up to the nearest multiple of 8 so that protocol messages will always be whole bytes.

Length of OPRF outputs.

The length of OPRF output controls the probability of a spurious collision in the PSI protocol. In Section 5.2 we argued that output length of $\sigma + \log_2(n^2)$ is sufficient to bound the probability of any spurious collision to $2^{-\sigma}$.

Using $\sigma = 40$, we compute the appropriate length in column v of Table 2. We round up to the nearest multiple of 8 so that protocol messages will always be whole bytes.

6.2 Environment settings

All of our experiments were implemented on a server with Intel(R) Xeon(R) CPU E5-2699 v3 2.30GHz CPU and 256 GB RAM. We run both clients on the same machine, but simulate a LAN and WAN connection using the Linux `tc` command. In the WAN setting, the average network bandwidth and the average (round-trip) latency are set to be 50 MB/s and 96 ms, respectively. In the LAN setting, the network has 0.2ms latency. All of our experiments use a single thread for each party.

6.3 Implementation Details

In our BaRK-OPRF protocol, the offline phase is conducted to obtain an OT extension matrix of size $(1.2n+s) \times k$ by using the IKNP OT extension. Specifically, first we use the Naor-Pinkas construction [21] to get 128 base-OTs, which are then extended to a $k \times 128$ matrix by utilizing the pseudorandom generator. The transpose of this matrix yields the k base OTs for the BaRK-OPRF extension protocol. We extend to $1.2n + s$ OPRF instances.

We hash all inputs of both client and server at the beginning of the online phase. Following Lemma 1, we use a PRF with suitably long output as our pseudorandom code. More concretely, the parties agree on an AES-128 key sk, which is independent of their inputs, and then extend the output of AES via:

$$C(x) = AES_{sk}(1\|x)\|AES_{sk}(2\|x)\|AES_{sk}(3\|x)\|AES_{sk}(4\|x)$$

to obtain the desired k random output bits. Furthermore, to reduce the waiting time at the server side, the client will constantly send a new packet encompassing multiple code words to the server. Based on trail-and-error approach, the packet size of $2^{12} \times k$ bits is selected to minimize the waiting time. In Table 4, we report the running time of our protocol for both offline and online phases in different settings. For instance, in LAN environment, the online phase of our BaRK-OPRF protocol takes about 3.2s for $n = 2^{20}$.

To illustrate the efficacy of the BaRK-OPRF-PSI approach, we compared it with a **naïve hashing** protocol and the **PSSZ** protocol. The naïve hashing protocol is a widely-used insecure protocol [27] where both parties use the same cryptographic hash function to hash their elements, then one of the parties permutes their hash value and sends the result to the other party, who will compute the intersection by computing the match of the hash values. In the following, we conducted several performance tests with the input sets of equal size n and for inputs of length 32, 64, and 128 bits.

Note that the running time of our PSI protocol does not depend on the bit length of the input. It can be explained as follows. First, the upper bound of the length of the input is 128 bits. Second, the hash function will call a block of 128 bits to encrypt the input data, thus our protocol has the same computation cost for all bit length of the input. In addition, the communication cost of our BaRK-OPRF protocol depends only on the length of the pseudorandom code k and the length v of the OPRF outputs, which are independent of the bit length ℓ. Similarly, the naïve hashing protocol does not depend on ℓ. This was confirmed by our simulation results for different bit lengths (e.g. 32 bits, 64 bits, and 128 bits). Table 3 presents the running time of the naïve hashing protocol, PSSZ, and our PSI protocol in both LAN and WAN environment.

As we can see in the tables, our protocol outperforms PSSZ in almost all the case studies, especially for the long bit length of input ℓ and large values of the input size n. For example, we consider the results in the LAN setting. For the input size of 2^{20}, our approach can improve 2.8 times and 3.6 times the performance of PSSZ for the bit lengths of 64 bits and 128 bits, respectively. For the input size of 2^{24}, the corresponding improvements are 2.3 times and 3.6 times. It is worth mentioning that it takes about 1 minute to compute the intersection for the sets of size $n = 2^{24}$. Similar observations can be inferred from Table 3 for the WAN setting.

At the same time, for smaller bit lengths, the PSSZ protocol can be faster than our PSI protocol. This is the case, for example, when the bit length is 32 bits and $n = 2^{24}$ in LAN setting. Since the two protocols are very similar, differing only in the choice of OPRF subprotocol, it would

Setting	Protocol	Bit length ℓ	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}
LAN	(insecure) naïve hashing	$\{32, 64, 128\}$	1	6	75	759	13,529
	PSSZ	32	306	380	770	4,438	**42,221**
		64	306	442	1,236	10,501	137,383
		128	307	443	1,352	13,814	213,597
	BaRK-OPRF-PSI	$\{32, 64, 128\}$	**192**	**211**	**387**	**3,780**	58,567
WAN	(insecure) naïve hashing	$\{32, 64, 128\}$	97	101	180	1,422	22,990
	PSSZ	32	609	701	1,425	8,222	**81,234**
		64	624	742	2,142	18,398	248,919
		128	624	746	2,198	23,546	381,913
	BaRK-OPRF-PSI	$\{32, 64, 128\}$	**556**	**585**	**1,259**	**7,455**	106,828

Table 3: Running time in ms for PSI protocols with n elements per party

Setting	Phase	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}
LAN	Offline	171	171	216	601	7,615
	Online	21	40	171	3,179	50,952
WAN	Offline	291	313	316	758	7,482
	Online	265	272	943	6,697	99,346

Table 4: Running time of our BaRK-OPRF protocol in ms in offline and online phases

Protocol	Bit length ℓ	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}	Asymptotic [bit]
naïve hashing	$\{32, 64, 128\}$	0.01	0.03	0.56	10.00	176.00	nv
PSSZ	32	0.06	0.77	9.18	142.80	1,574.40	$2\kappa(1.2n+s)\lceil\frac{\min(v',\ell)-\log(n)}{8}\rceil+(3+s)nv'$
	64	0.09	1.37	18.78	296.40	4,032.00	
	128	0.10	1.52	23.58	411.60	6,489.60	
BaRK-OPRF-PSI	$\{32, 64, 128\}$	0.04	0.53	8.06	127.20	1,955.20	$k(1.2n+s)+(3+s)nv$

Table 5: Communication in MB for PSI protocols with n elements per party. Parameters k, s, and v refer to those in Table 2 / Section 6.1. PSSZ requires slightly long OPRF outputs: $v' = \sigma + \log(3n^2)$. Communication costs for PSSZ and for our protocol ignore the fixed cost of base OTs for OT extension.

be relatively straightforward to implement a hybrid that always chooses the best OPRF subprotocol based on n and ℓ according to Table 1. However, in order to clarify the strengths/weaknesses of the two protocols, we report the performance for our approach even when it is worse.

Similar to the running time result, our communication cost is 2.9–3.3× faster than Pinkas et al. for PSI of 128-bit strings and sufficiently large sets. Concretely, for the input size of 2^{20}, our protocol can improve 3.2 times the performance of PSSZ for the bit lengths 128 bits. Table 5 presents the communication (in MB) of the naïve hashing protocol, PSSZ, and our BaRK-OPRF-PSI protocol.

Acknowledgments

We thank Peter Rindal for contributing libraries and helpful suggestions to our protocol implementation. We also thank Michael Zohner for answering our many questions about the implementation of [27]. Finally, we thank the anonymous CCS reviewers for their helpful feedback.

The first author is supported by the Office of Naval Research (ONR) contract number N00014-14-C-0113. The second author is supported by NSF Grants CNS-1350619 and CNS-1414119, in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226, and an MIT Translational Fellowship. The third and fourth authors are supported by NSF award 1149647 and a Google research award. This work was initiated while the first three authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

7. REFERENCES

[1] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer and extensions for faster secure computation. In Sadeghi et al. [29], pp. 535–548.

[2] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT 2015, Part I* (Sofia, Bulgaria, Apr. 26–30, 2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *LNCS*, Springer, Heidelberg, Germany, pp. 673–701.

[3] Camenisch, J., Neven, G., and shelat, a. Simulatable adaptive oblivious transfer. In *EUROCRYPT 2007* (Barcelona, Spain, May 20–24, 2007), M. Naor, Ed., vol. 4515 of *LNCS*, Springer, Heidelberg, Germany, pp. 573–590.

[4] Dodis, Y., and Yampolskiy, A. A verifiable random function with short proofs and keys. In *PKC 2005* (Les Diablerets, Switzerland, Jan. 23–26, 2005), S. Vaudenay,

Ed., vol. 3386 of *LNCS*, Springer, Heidelberg, Germany, pp. 416–431.

[5] Faust, S., Hazay, C., and Venturi, D. Outsourced pattern matching. In *ICALP 2013, Part II* (Riga, Latvia, July 8–12, 2013), F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, Eds., vol. 7966 of *LNCS*, Springer, Heidelberg, Germany, pp. 545–556.

[6] Freedman, M. J., Ishai, Y., Pinkas, B., and Reingold, O. Keyword search and oblivious pseudorandom functions. In *TCC 2005* (Cambridge, MA, USA, Feb. 10–12, 2005), J. Kilian, Ed., vol. 3378 of *LNCS*, Springer, Heidelberg, Germany, pp. 303–324.

[7] Goldreich, O. *Foundations of Cryptography, Volume 2: Basic Applications*. Cambridge University Press, The address, 2004.

[8] Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC* (New York City,, New York, USA, May 25–27, 1987), A. Aho, Ed., ACM Press, pp. 218–229.

[9] Hazay, C. Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs. In *TCC 2015, Part II* (Warsaw, Poland, Mar. 23–25, 2015), Y. Dodis and J. B. Nielsen, Eds., vol. 9015 of *LNCS*, Springer, Heidelberg, Germany, pp. 90–120.

[10] Hazay, C., and Lindell, Y. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of Cryptology 23*, 3 (July 2010), 422–456.

[11] Huberman, B. A., Franklin, M. K., and Hogg, T. Enhancing privacy and trust in electronic communities. In *EC* (1999), pp. 78–86.

[12] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. In *CRYPTO 2003* (Santa Barbara, CA, USA, Aug. 17–21, 2003), D. Boneh, Ed., vol. 2729 of *LNCS*, Springer, Heidelberg, Germany, pp. 145–161.

[13] Jarecki, S., and Liu, X. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC 2009* (Mar. 15–17, 2009), O. Reingold, Ed., vol. 5444 of *LNCS*, Springer, Heidelberg, Germany, pp. 577–594.

[14] Keller, M., Orsini, E., and Scholl, P. Actively secure OT extension with optimal overhead. In *CRYPTO 2015, Part I* (Santa Barbara, CA, USA, Aug. 16–20, 2015), R. Gennaro and M. J. B. Robshaw, Eds., vol. 9215 of *LNCS*, Springer, Heidelberg, Germany, pp. 724–741.

[15] Kilian, J. Founding cryptography on oblivious transfer. In *20th ACM STOC* (Chicago, Illinois, USA, May 2–4, 1988), ACM Press, pp. 20–31.

[16] Kolesnikov, V. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT 2005* (Chennai, India, Dec. 4–8, 2005), B. K. Roy, Ed., vol. 3788 of *LNCS*, Springer, Heidelberg, Germany, pp. 136–155.

[17] Kolesnikov, V., and Kumaresan, R. Improved secure two-party computation via information-theoretic garbled circuits. In *SCN 12* (Amalfi, Italy, Sept. 5–7, 2012), I. Visconti and R. D. Prisco, Eds., vol. 7485 of *LNCS*, Springer, Heidelberg, Germany, pp. 205–221.

[18] Kolesnikov, V., and Kumaresan, R. Improved OT extension for transferring short secrets. In *CRYPTO 2013, Part II* (Santa Barbara, CA, USA, Aug. 18–22, 2013), R. Canetti and J. A. Garay, Eds., vol. 8043 of *LNCS*, Springer, Heidelberg, Germany, pp. 54–70.

[19] Kolesnikov, V., and Malozemoff, A. J. Public verifiability in the covert model (almost) for free. In *ASIACRYPT 2015, Part II* (Auckland, New Zealand, Nov. 30 – Dec. 3, 2015), T. Iwata and J. H. Cheon, Eds., vol. 9453 of *LNCS*, Springer, Heidelberg, Germany, pp. 210–235.

[20] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 20–20.

[21] Naor, M., and Pinkas, B. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2001), SODA '01, Society for Industrial and Applied Mathematics, pp. 448–457.

[22] Naor, M., and Reingold, O. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM 51*, 2 (2004), 231–262.

[23] Nielsen, J. B. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. ia.cr/2007/215.

[24] Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S. A new approach to practical active-secure two-party computation. In *CRYPTO 2012* (Santa Barbara, CA, USA, Aug. 19–23, 2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *LNCS*, Springer, Heidelberg, Germany, pp. 681–700.

[25] Opsahl, K. The disconcerting details: How Facebook teams up with data brokers to show you targeted ads. https://www.eff.org/deeplinks/2013/04/disconcerting-details-how-facebook-teams-data-brokers-show-you-targeted-ads, 2013. [Online; accessed 23-May-2016].

[26] Pagh, R., and Rodler, F. F. Cuckoo hashing. *J. Algorithms 51*, 2 (2004), 122–144.

[27] Pinkas, B., Schneider, T., Segev, G., and Zohner, M. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium, USENIX Security 15* (2015), J. Jung and T. Holz, Eds., USENIX Association, pp. 515–530.

[28] Pinkas, B., Schneider, T., and Zohner, M. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium, USENIX Security 14* (2014), K. Fu and J. Jung, Eds., USENIX Association, pp. 797–812.

[29] Sadeghi, A.-R., Gligor, V. D., and Yung, M., Eds. *ACM CCS 13* (Berlin, Germany, Nov. 4–8, 2013), ACM Press.

[30] Yao, A. C.-C. How to generate and exchange secrets (extended abstract). In *27th FOCS* (Toronto, Ontario, Canada, Oct. 27–29, 1986), IEEE Computer Society Press, pp. 162–167.

[31] Yung, M. From mental poker to core business: Why and how to deploy secure computation protocols? https://www.sigsac.org/ccs/CCS2015/pro_keynote.html, 2015. ACM CCS 2015 Keynote Talk.