QuietOT: Lightweight Oblivious Transfer with a Public-Key Setup

Geoffroy Couteau, Lalita Devadas, Srinivas Devadas, Alexander Koch, Sacha Servan-Schreiber

QuietOT: Lightweight Oblivious Transfer with a Public-Key Setup

Geoffroy Couteau[1,2], Lalita Devadas[3], Srinivas Devadas[3], Alexander Koch[1,2], and Sacha Servan-Schreiber[3]

[1] CNRS
[2] IRIF, Université Paris Cité
[3] MIT

Abstract. Oblivious Transfer (OT) is at the heart of secure computation and is a foundation for many applications in cryptography. Over two decades of work have led to extremely efficient protocols for evaluating OT instances in the preprocessing model, through a paradigm called OT extension. A few OT instances generated in an offline phase can be used to perform many OTs in an online phase efficiently, i.e., with very low communication and computational overheads.

Specifically, traditional OT extension protocols use a small number of "base" OTs, generated using any black-box OT protocol, and convert them into many OT instances using only lightweight symmetric-key primitives. Recently, a new paradigm of OT with a *public-key setup* has emerged, which replaces the base OTs with a non-interactive setup: Using only the public key of the other party, two parties can efficiently compute a virtually unbounded number of OT instances on-the-fly.

In this paper, we put forth a novel framework for OT extension with a public-key setup (henceforth, "public-key OT") and concretely efficient instantiations. Implementations of our framework are 30–100× faster when compared to the previous state-of-the-art public-key OT protocols, and remain competitive even when compared to OT protocols that *do not* offer a public-key setup. Additionally, our instantiations result in the first public-key schemes with plausible post-quantum security.

In summary, this paper contributes:

- QuietOT: A framework for OT extension with a public-key setup that uses fast, symmetric-key primitives to generate OT instances following a one-time public-key setup, and offering additional features such as precomputability.
- A public-key setup for QuietOT from the RingLWE assumption, resulting in the first post-quantum construction of OT extension with a public-key setup.
- An optimized, open-source implementation of our construction that can generate up to 1M OT extensions per second on commodity hardware. In contrast, the state-of-the-art public-key OT protocol is limited to approximately 20K OTs per second.
- The first formal treatment of the security of OT with a public-key setup in a multi-party setting, which addresses several subtleties that were overlooked in prior work.

Table of Contents

1 Introduction

In its simplest form, Oblivious Transfer (OT) allows a party (called the *receiver*) to privately retrieve one out of two messages from another party (called the *sender*). The receiver has a choice bit b and the sender has a pair of messages (m_0, m_1). Using OT, the receiver learns m_b but learns nothing about m_{1-b}. Moreover, the sender is guaranteed to learn nothing about b. OT is a foundational building block for secure multiparty computation [57], and its applications typically require a large number of oblivious transfers (in the millions or billions). Unfortunately, all existing protocols for OT require public-key cryptography, making them concretely inefficient in many applications. Since it is known that OT cannot be constructed in a black-box manner using only symmetric-key primitives [51], this inefficiency is somewhat inherent to the OT problem. Fortunately, however, since the seminal work of Beaver [14] and the efficient construction of Ishai, Kilian, Nissim, and Petrank [53] (henceforth, IKNP), lightweight OT can be realized by performing a small number of expensive "base" OTs that are then extended (using only lightweight, symmetric-key cryptography) or "reused" to perform any number of regular OTs. Despite its concrete computational efficiency, the original paradigm of IKNP induces a large communication overhead (λ bits of communication per extended OT). To address this overhead, new paradigms have recently emerged that enable extending OTs using much less communication. Protocols like SoftSpoken OT [69] directly improve the communication efficiency of IKNP by a small factor k (e.g., $k = 5$) at the cost of some increased computation. Silent OT extension protocols [63, 22, 24, 21, 67, 70, 75, 38] achieve optimal communication (3 bits of communication per OT), but come with a concrete computational overhead that is noticeably larger than SoftSpoken OT (e.g., RRT, the state-of-the-art silent OT [67], being about 8× slower than SoftSpoken OT on machines with AVX instructions).

Our goal: "Diffie-Hellman" for Secure Computation. The state-of-the-art techniques for efficiently evaluating a large number of OT instances all require the sender and the receiver to initially interact in a distributed setup phase. Contrast this with the simpler task of establishing a secure *communication* channel on the Internet. Thanks to the breakthrough key-agreement protocol of Diffie and Hellman [42] in 1976, any pair of parties can locally derive a shared symmetric encryption key directly from the *public key* of the other party. This approach to securing communication has proven to be highly effective, and is now widely deployed [72]. Concretely, this means that over a large network of N parties, all pairs of parties can securely communicate following a one-time public-key setup with $O(N)$ communication, where all parties broadcast their public keys.

The goal of *oblivious transfer with a public-key setup*, first explicitly put forth by Orlandi, Scholl, and Yakoubov [63], is to achieve a similar feature for the task of secure *computation* on the Internet. Concretely, over a large network of N parties, if all pairs of parties want to be able to jointly run secure *computation* protocols (which typically requires evaluating many OTs), they must all run the distributed setup pair-wise, resulting in $O(N^2)$ communication and simultaneous interactions. However, with a *public-key setup* (or non-interactive "public-key OT" for short), each pair of parties can instead efficiently generate an arbitrary number of pseudorandom OT instances, given only the public key of the other party! These pseudorandom OT instances can then be derandomized in one round to perform a regular OT [63, 30, 15]. Unfortunately, despite being very desirable, this feature is not achievable with any of the state-of-the-art OT extension protocols, even in the semi-honest model.

Very recently, however, the work of Bui, Couteau, Meyer, Passelègue, and Riahinia [30] (henceforth, BCMPR) achieved the first practically efficient candidate construction of public-key OT by building a new Pseudorandom Correlation Function (PCF) for the OT correlation, and showing that it admits a public-key setup. Concretely, with a public-key PCF, two parties can, given only each other's public key, locally generate an arbitrary amount of pseudorandom OTs. In turn, these pseudorandom OTs can be used to perform a regular bit-OT in one round of interaction and three bits of communication). While this represents significant progress, their result falls short of providing a fully satisfactory solution to the problem of efficient public-key OT. For one, their protocol is *not* an OT extension, given that it requires (local) public-key operations for *every* OT that it generates. Consequently, it is considerably less efficient than state-of-the-art OT extension protocols. Concretely, BCMPR can generate up to 21K OTs per second, whereas state-of-the-art OT extension protocols can generate several million OTs per second [69]. Additionally, BCMPR is built around group-based primitives, making it not post-quantum secure, and relies on a new assumption they call "Sparse-power DDH" (or SPDDH for short) which is only proven secure in the generic group model.

1.1 Our contributions

In this paper, we make several contributions, which we highlight here and describe in depth in our technical overview of Section 2. The primary contribution of this paper is QuietOT: a novel framework for fast OT extension with a public-key setup. With QuietOT, given only each other's public key, two parties can generate an arbitrary amount of pseudorandom "*ListOTs*," a variant of OT which we introduce, which can be converted into pseudorandom (resp. regular) OTs in one round and a small overhead in communication, e.g., using 4 bits/OT (resp. 7 bits/OT) in one of our instantiations. The only difference between our approach via ListOT and a standard PCF for OT is that the derandomization step incurs slightly more communication (e.g., 7 bits instead of 3 bits). Unlike all prior public-key OT protocols, QuietOT does *not* require public-key operations when generating OTs, making the concrete performance *one to six orders* of magnitude faster compared to the state-of-the-art OT protocols that offer a public-key setup. Moreover, the public-key setup of QuietOT can be replaced by a simple two-round setup using any black-box base OTs, yielding new constructions of two-round OT extension. Additionally, we show that the base OTs can be replaced with a public-key setup under the standard RingLWE assumption (with a superpolynomial modulus-to-noise ratio), allowing parties to non-interactively derive a shared key from which they can generate OT extensions.

We note that state-of-the-art OT extension protocols, such as SoftSpoken [69], remain significantly faster than QuietOT (e.g., about 7× faster in the regime where SoftSpoken communicates 16 bits/OT). The core advantage of QuietOT over these alternatives lies in its public-key setup: concretely, using QuietOT, two parties can execute the vast majority of the computation *before they even interact*, given only each other's public key. The interactive phase that follows involves solely cheap, non-cryptographic operations (a few XORs per OT). In contrast, using SoftSpoken or any state-of-the-art OT extension, the parties must *first* interact (to generate base OTs) *before* they can run the bulk of the computation and interact again to complete the protocol; this can cause significant delays during which both parties have to stay online. We believe that this precomputation feature of QuietOT is highly desirable in the setting of on-demand pairwise secure computation over a large network. As a bonus, QuietOT communicates less than SoftSpoken, and requires less rounds of interaction.

Under the hood, our framework combines any "Inner-Product Membership" weak PRF (IPM-wPRF) [30] with a Shiftable Constrained Pseudorandom Function (ShCPRF), a new primitive that we introduce in Section 5. Prior work [30, 63] requires using public-key operations for each OT, translating to expensive group operations under either the Quadratic Residuosity (QR) or DDH assumption (the construction of Orlandi, Scholl, and Yakoubov [63], henceforth OSY, is mostly of theoretical interest due to the large number of group exponentiations required). In contrast, we show that a ShCPRF can be constructed unconditionally in the random oracle model by exploiting a recent CPRF construction [71]. In addition, because IPM-wPRF are lightweight symmetric-key primitives (i.e., they do not require public-key operations to evaluate), our overall OT extension protocol is very efficient. We provide a comparison to related work in Table 1.

Additional contributions. In addition to the main contribution of the QuietOT framework, this paper contributes:

- The first formal treatment of public-key OT when used in a secure multi-party computation over a large network. Our definitions and analysis address several subtle issues with using any public-key OT constructions (including BCMPR and OSY) in a multi-party setting where security must be guaranteed with respect to an adversary corrupting a subset of parties.
- A definition and construction of shiftable CPRFs, a twist on CPRFs where the master key holder can efficiently "shift" the constraint when evaluating the CPRF. This construction plays a crucial role in our framework and may be of independent interest.
- An open-source, optimized implementation of the BCMPR protocol (Bui et al. [30] do not provide an implementation of their public-key PCF), which we evaluate and compare to QuietOT in Section 8.

2 Technical Overview

In this section, we provide a detailed overview of our results. In Section 2.1, we start by covering the state-of-the-art BCMPR framework for public-key OT. Then, in Section 2.2, we cover the main

	OT/s Max. Throughput	Bits/OT Communication	PKS†	PQ	Assumptions
IKNP	34,000,000	128	✗	✓	ROM
SoftSpoken ($k = 2$)	53,000,000	64	✗	✓	ROM
SoftSpoken ($k = 8$)	9,500,000	16	✗	✓	ROM
RRT	6,900,000	3	✗	✓	EC-LPN+ROM
OSY	1	3	✓	✗	QR+ROM
BCMPR	21,000	3	✓	✗	IPM-wPRF+SPDDH+ROM
QuietOT	1,200,000	7	✓	✓	IPM-wPRF+ROM

Table 1: An overview of OT extension protocols and their maximum throughput observed across different hardware and parameter settings (full evaluation results provided in Section 8). PKS and PQ indicate whether the construction has a *public-key setup* and is plausibly *post-quantum* secure, respectively. For efficiency, nearly all OT extension protocols are instantiated in the Random Oracle Model (ROM). Note that in these constructions, the random oracle assumption can be generically replaced with a suitable correlation-robust hash function. †PKS not implemented.

ideas behind our QuietOT framework and the different realizations of it. In Section 2.3, we show how QuietOT implies two-round OT extension and full pre-computability for either the sender or the receiver. In Section 2.4, we explain our approach to non-interactive public-key setup under the RingLWE assumption. In Section 2.5, we overview our definitions of public-key setup with *multi-instance* security, which becomes a crucial building block for applying public-key OT to a multi-party computation setting.

2.1 Background on the BCMPR framework

The BCMPR framework constructs a pseudorandom correlation function (PCF) for OT correlations using an IPM-wPRF. In addition to the IPM-wPRF requirement, they also require the Sparse-Power DDH (SPDDH) assumption and instantiate their public-key setup from the DCR assumption. In their PCF construction, the sender and receiver can compute OT correlations on-demand: The sender outputs two pseudorandom bits (s_0, s_1) while the receiver outputs (b, s_b), where $b \in \{0,1\}$ is a pseudorandom choice bit. This correlation can then be converted into a chosen-bit OT with 3 bits of communication in one round of interaction using the transformation of Beaver [15].

At the heart of the BCMPR framework is a Constrained PRF (CPRF) $F = (F.\mathsf{KeyGen}, F.\mathsf{Eval}, F.\mathsf{Constrain}, F.\mathsf{CEval})$ with the constraint predicate set to a weak PRF[4] $f_{\mathbf{z}}$ that outputs a pseudorandom bit. At a high level, a CPRF has two keys: a master key and a constrained key. The constrained key only allows evaluating the PRF when the constraint predicate is satisfied. Hence, the weak PRF f indicates if the input to F is constrained or not, making roughly half the inputs to F constrained. It was known (somewhat folklore) that any CPRF with a weak PRF as a constraint predicate can be used to construct a PCF for OT correlations [13]. However, prior to BCMPR, all existing CPRF constructions were either not sufficiently expressive to evaluate a weak PRF as a predicate or not concretely efficient enough to result in practical solutions [31, 27, 33, 64, 28, 37, 9]. Therefore, at the core of BCMPR is a construction of a CPRF *just* powerful enough to evaluate a suitable weak PRF candidate as the constraint predicate while remaining concretely efficient in practice. They realize such a CPRF by adapting the classical Naor-Reingold PRF [62]. In a nutshell, the BCMPR framework for OT correlations combines:

- A CPRF supporting a special class of *Inner-Product Membership* (IPM) constraints (the constrained key can evaluate the PRF on x if and only if $\langle \mathbf{z}, x \rangle \in S$, for some constraint vector $\mathbf{z} \in \mathcal{R}^n$ defined over a finite ring $\mathcal{R}$ and fixed set S partitioning the ring elements)[5], and
- Any weak PRF $f_{\mathbf{z}} : \{0,1\}^n \to \{0,1\}$ having an evaluation function that can be described as an inner-product membership predicate (which they call an IPM-wPRF). That is, $f_{\mathbf{z}}(x) =$

[4] A weak PRF is only pseudorandom on *uniformly random* inputs.
[5] We slightly abuse notation by interpreting the bit string x as a *vector* of bits.

0 iff $\langle \mathbf{z}, x \rangle \in S_0$ and $f_{\mathbf{z}}(x) = 1$ iff $\langle \mathbf{z}, x \rangle \in S_1$, for a partitioning $S_0 \cup S_1$ of the finite ring $\mathcal{R}$ over which the inner product is defined, and a vector $\mathbf{z} \in \mathcal{R}^n$.

Building on the Naor-Reingold PRF, BCMPR constructs a CPRF supporting IPM predicates in the Random Oracle Model (ROM) [17]. In particular, using any IPM-wPRF (which can be realized from a handful of assumptions), coupled with their CPRF construction supporting IPM predicates, allows them to instantiate the following generic template for building a PCF for OT correlations, which will serve as inspiration for our framework as well.

A general PCF template from CPRFs for IPM predicates. The template of BCMPR uses a CPRF F with IPM constraints that evaluates an IPM-wPRF $f_{\mathbf{z}}$ as the predicate, for a vector $\mathbf{z} \in \mathcal{R}^n$ that we will call the wPRF key. The sender gets two master keys $(\mathsf{msk}_0, \mathsf{msk}_1)$ for F, while the receiver obtains two constrained keys $(\mathsf{csk}_0, \mathsf{csk}_1)$. The master keys can be used to evaluate the PRF on the entire domain. In contrast, the constrained key can only be used to evaluate the PRF when the constraint predicate is satisfied. The idea is to have the constrained key csk_0 have $f_{\mathbf{z}}$ as the predicate, and csk_1 have the opposite predicate $1 - f_{\mathbf{z}}$. Notice that given the two constrained keys, the receiver can only evaluate the CPRF using *one* of the two keys for an input x (depending on the value of $f_{\mathbf{z}}(x)$, which is pseudorandom). Moreover, given the IPM-wPRF key $\mathbf{z}$, the receiver can determine which of the two evaluations is constrained for an input x by evaluating the "predicate" $f_{\mathbf{z}}(x)$. The receiver can then compute and output the correlation (b, s_b), consisting of the pseudorandom bit $b = f_{\mathbf{z}}(x)$ and the string $s_b = F.\mathsf{CEval}(\mathsf{csk}_b, x)$. The sender, in contrast, only obtains the master keys $(\mathsf{msk}_0, \mathsf{msk}_1)$, which are independent of the IPM-wPRF key $\mathbf{z}$. As such, the sender can only compute the strings $s_0 = F.\mathsf{Eval}(\mathsf{msk}_0, x)$ and $s_1 = F.\mathsf{Eval}(\mathsf{msk}_1, x)$, consisting of the sender's correlation (s_0, s_1), without learning the pseudorandom bit b computed by the receiver.

Limitations of the general template. The core difficulty associated with the above template (and the BCMPR framework by extension) is finding a CPRF with a predicate class that is sufficiently powerful to evaluate $f_{\mathbf{z}}$. The most efficient construction to date is the constrained Naor-Reingold PRF construction of BCMPR, which (1) requires a new cryptographic assumption, (2) is not post-quantum secure and, (3) necessitates concretely expensive group operations to evaluate, placing an upper limit on practical efficiency of BCMPR (e.g., 21K correlations per second in our optimized implementation). In contrast, OT extension protocols like SoftSpoken OT [69] (which generalizes IKNP) use only lightweight symmetric-key primitives, are post-quantum secure, and are incredibly fast (e.g., achieving several million correlations per second), but do not offer public-key setups. Unfortunately, improving the efficiency of the BCMPR framework hinges on developing more efficient CPRF constructions for IPM predicates, which appears to be the weakest class of predicates sufficiently powerful to evaluate any wPRF. Note that a pseudorandom function *cannot* have a linear evaluation, and therefore inner-product equality predicates are inherently insufficient.

2.2 Our approach

Intuition. The starting point of our approach is the template construction of BCMPR. As with BCMPR, in our framework, the receiver holds the key $\mathbf{z} \in \mathcal{R}^n$ of an IPM-wPRF and we let the (pseudorandom) selection bit of the receiver be defined as the output $f_{\mathbf{z}}(x)$ of the wPRF f on a random input x. Recall that $f_{\mathbf{z}}(x) = b$ iff $\langle \mathbf{z}, x \rangle \in S_b$ (where S_0, S_1 are a public partitioning of the inner-product range, associated with the IPM-wPRF). The main limitation of BCMPR is their reliance on a CPRF for a class of constraints that contains $f_{\mathbf{z}}$. While they provide an optimized construction, it still requires public-key operations (group exponentiation) for *every* evaluation, and hence for every OT instance.

At this point, we diverge significantly from the BCMPR framework by replacing their CPRF with a far more efficient primitive. Our starting point is a recent CPRF construction of Servan-Schreiber [71], which uses only symmetric-key primitives. Concretely, evaluating the CPRF involves computing an inner product and hashing the result; furthermore, the CPRF was shown to be unconditionally secure in the ROM. However, the catch is that the CPRF of Servan-Schreiber only handles inner-product predicates: That is, given a constraint $\mathbf{z}$, the constrained evaluation with csk on x matches the evaluation with the master key if and only if $\langle \mathbf{z}, x \rangle = 0 \in \mathcal{R}$. Observe that using this much weaker CPRF, the receiver is now only able to evaluate F on all inputs where $\langle \mathbf{z}, x \rangle = 0$[6] (roughly $\frac{1}{|S_b|}$ of

[6] We follow the convention of letting $P(x) = 0$ when the predicate P is satisfied.

all inputs assuming $f(x) = b$, and where $|S_b| \approx |\mathcal{R}|/2$), which is too weak to instantiate the BCMPR template.

Shiftable CPRFs to the rescue. Our first key observation is that (a slight modification of) the CPRF framework of Servan-Schreiber enjoys an additional *shiftability* property. Concretely, the CPRF evaluation with the master key msk can take an additional *shift* α as input, and provides the following guarantee: The constrained evaluation $F.\mathsf{CEval}(\mathsf{csk}, x)$ is equal to $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ whenever $\langle \mathbf{z}, x \rangle - \alpha = 0$. That is, the constraint is *shifted by* α. Given such a shiftable CPRF for inner products, the sender can now compute $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ for *all possible shifts* $\alpha \in S_0 \cup S_1$. This yields two lists of values: $L_0 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_0}$ and $L_1 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_1}$. Our next core observation is that the value $F.\mathsf{CEval}(\mathsf{csk}, x)$ computed by the receiver belongs to exactly one of the two lists, and furthermore, *the index b of the list L_b it belongs to is simply $f_{\mathbf{z}}(x)$*. That is, the receiver knows a pseudorandom value v and pseudorandom "selection bit" $b = f_{\mathbf{z}}(x)$ such that $v \in L_b$. Additionally, by using the constraint $\mathbf{z}$, the receiver can determine the index i in L_b in which v is located (i.e., such that $v = L_b[i]$).

Oblivious transfer from ListOT. So far, we have seen that given a shiftable CPRF for inner-product predicates, the sender and the receiver can generate many instances of the following "correlation:" The sender gets as output two (pseudorandom) lists (L_0, L_1), and the receiver obtains (v, b, i) where $v = L_b[i]$, and b is pseudorandom from the viewpoint of the sender (however, importantly, i is *not* pseudorandom, which prevents this from being a true OT correlation). We call "ListOT" this weaker variant of the OT correlation. The name is inspired from *list decoding* [44], where a decoding algorithm for a code is allowed to output a list of code words from which the word can be decoded.[7] Hence, for ListOT, the sender outputs two lists of messages, L_0, L_1, and the receiver outputs a bit b, value v, *and* an index key i, such that v is located at $L_b[i]$. (Later, for ease of notation, L_0 and L_1 will be treated as key-value stores/dictionaries.)

While the pseudorandom ListOT instances are *not* correlations in the strict technical sense (because the distribution of i depends on the secret wPRF key), it is not too hard to see that it still suffices to instantiate a random OT using some additional communication. To see this, observe that given OT inputs (m_0, m_1), the sender simply sends $(L_0[j] \oplus m_0)_{j \in S_0}$ and $(L_1[j] \oplus m_1)_{j \in S_1}$ to the receiver. The receiver recovers m_b by unmasking $L_b[i] \oplus m_b$ using $v = L_b[i]$.[8]

We now explain how we construct efficient ShCPRFs by adapting the framework of Servan-Schreiber [71] building CPRFs for inner-product predicates from RKA-secure PRFs [18] in the standard model (or in the random oracle model).

Constructing ShCPRFs. We make the observation that in all existing CPRF constructions for inner-product predicates [71, 30, 40], the master key holder can efficiently compute the set of all possible pseudorandom values evaluated under the constrained key csk. We will focus on the CPRF construction of Servan-Schreiber, instantiated unconditionally using a hash function H modeled as a random oracle. In this construction, the master key msk consists of a random vector $\mathbf{z_0}$ of length n, with elements from some sufficiently large field $\mathbb{F}$.[9] For a constraint vector $\mathbf{z} \in \mathbb{F}^n$, the constrained key is defined as $\mathbf{z_1} = \mathbf{z_0} - \Delta \cdot \mathbf{z}$, where $\Delta \in \mathbb{F} \setminus \{0\}$ is random. Simplifying slightly,[10] the evaluation and the constrained evaluation algorithms are defined as $H(\langle \mathbf{z_0}, x \rangle, x)$ and $H(\langle \mathbf{z_1}, x \rangle, x)$, respectively. Note that when $\langle \mathbf{z}, x \rangle = 0$, it holds that $H(\langle \mathbf{z_0}, x \rangle, x)$ is equal to $H(\langle \mathbf{z_1}, x \rangle, x)$, which guarantees the master key and constrained key evaluations agree. In contrast, when $\langle \mathbf{z}, x \rangle \neq 0$, then $H(\langle \mathbf{z_1}, x \rangle, x)$ is equal to $H(\langle \mathbf{z_0}, x \rangle - \Delta \langle \mathbf{z}, x \rangle, x)$, which is independent of $H(\langle \mathbf{z_0}, x \rangle, x)$ due to Δ. In particular, we observe that when $\langle \mathbf{z}, x \rangle \neq 0$, using $\mathbf{z_0}$ and Δ allows the master key holder to evaluate *all possible* constrained evaluations by computing $H(\langle \mathbf{z_1}, x \rangle + \Delta\alpha, x)$, for all possible inner products $\alpha \in \{\langle \mathbf{z}, x \rangle \mid x \in \{0,1\}^n\}$ associated with the constraint class given by $\mathbf{z}$. We point to Section 4 for more details on this ShCPRF construction. Abstractly, we define the master key evaluation algorithm $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ to take a shift α as an additional input while leaving the remaining CPRF algorithms unchanged.

[7] We note that ListOT is not related to "list two-party computation" [34], which defines list OT as a security definition for the standard oblivious transfer functionality.

[8] When generating (pseudo)random OTs, this simple approach can be further improved by letting m_0 and m_1 be the first element of L_0 and L_1 respectively, which allows communicating two elements less, for a total of $|S_0| + |S_1| - 2$ bits of communication. Concretely, with our most communication-efficient instance, this translates to only 4 bits of communication per random OT.

[9] Our actual ShCPRF construction is defined using a ring extension for efficiency.

[10] The full construction has an extra additive term to handle the all-zero input $x = 0^n$.

Putting things together: A "PCF" for ListOT. Using the ShCPRF construction sketched above, coupled with an IPM-wPRF $f_{\mathbf{z}}$ with partitioning $S_0 \cup S_1$, the sender with the master secret key msk computes the two lists, L_0 and L_1, corresponding to $b = 0$ and $b = 1$, respectively, as $L_0 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_0}$, $L_1 = (F.\mathsf{Eval}(\mathsf{msk}, x, \beta))_{\beta \in S_1}$, using a random x. Importantly, note that given the constrained key csk for a constraint vector $\mathbf{z}$, the receiver obtains *one* value in L_b, where $b = f_{\mathbf{z}}(x)$. All other values, in both lists, remain pseudorandom from the viewpoint of the receiver. At this stage, our framework can be instantiated using any choice of ShCPRF and any choice of IPM-wPRF. We choose to instantiate the ShCPRF in the random oracle model, as it offers the most concretely-efficient solution. The IPM-wPRF can be realized from several assumptions, as detailed in BCMPR. Indeed, many wPRFs fit the IPM-wPRF framework, including the Learning-with-Rounding (LWR)-based wPRF [11], the Goldreich-Applebaum-Raykov (GAR) [47, 6], and several other low-complexity wPRF candidates, including the Boneh, Ishai, Passelègue, Sahai, and Wu (BIPSW) [19], and LPN-based candidates [23]. (Bui et al. [30] provide an overview of these different candidates and others). The BIPSW wPRF candidate is especially well-suited to this framework given that the evaluation (defined in Equation (1)) is essentially just a rounded inner product computed in $\mathbb{Z}_6$:

$$f_{\mathbf{z}}(x) = \lfloor \langle \mathbf{z}, x \rangle \mod 6 \rceil_2 . \tag{1}$$

Note that when $f_{\mathbf{z}}(x) = 0$, then it holds that $\langle \mathbf{z}, x \rangle \pmod 6 \in \{0, 1, 2\}$ and when $f_{\mathbf{z}}(x) = 1$ it holds that $\langle \mathbf{z}, x \rangle \pmod 6 \in \{3, 4, 5\}$. By instantiating the ShCPRF to compute predicates over an extension of $\mathbb{Z}_6$, we can achieve incredibly efficient evaluations using the BIPSW IPM-wPRF (see Section 8 for our evaluation).

2.3 Two-round OT extension

Using our framework, we obtain a *two-round* OT extension protocol. Observe that the sender can independently generate the ShCPRF master secret key, consisting of $\mathbf{z_0}$ and Δ, while the receiver can independently generate the IPM-wPRF key $\mathbf{z}$. For the case where $\mathbf{z} \in \{0,1\}^n$, we can use any two-round string OT protocol repeated in parallel n times as follows. For $i \in [n]$, the sender sets $m_{i,0} = \mathbf{z_0}_i$ and $m_{i,1} = \mathbf{z_0}_i - \Delta$. The receiver uses $\mathbf{z}_i \in \{0,1\}$ as its choice bit to retrieve $m_{i,\mathbf{z}_i} = \mathbf{z_0}_i - \Delta \mathbf{z}_i$, and in this way can recover $\mathsf{csk} := \mathbf{z_0} - \Delta \mathbf{z}$ using n parallel calls to the two-round OT functionality (indeed, because Δ is the same across messages, any *correlated* OT protocol [7] is sufficient). In the general case, when $\mathbf{z} \in \mathcal{R}^n$, we can use any two round "reverse" Vector Oblivious Linear Evaluation (VOLE) protocol [3, 22], which directly generalizes correlated OT to work over a ring $\mathcal{R}$. In reverse VOLE, the sender inputs $(\mathbf{b}, x) \in \mathcal{R}^n \times \mathcal{R}$ and the receiver inputs $\mathbf{a} \in \mathcal{R}^n$. The sender receives no output while the receiver obtains $\mathbf{a}x + \mathbf{b}$. By letting the sender input $(\mathbf{z_0}, \Delta)$ and the receiver input $-\mathbf{z}$, we immediately have that the receiver obtains $\mathbf{z_1} = \mathbf{z_0} - \Delta \mathbf{z}$. See Appendix C for more details.

Two-round OT extension is known to be impossible under black-box symmetric-key primitives [45] making our use of an IPM-wPRF a rather weak assumption to circumvent the impossibility result of Garg et al. [45] (in fact, an IPM-PR**G** suffices). In contrast, protocols like IKNP and SoftSpoken inherently require three rounds of interaction due to their unconditional instantiations in the random oracle model, and all previous two-round OT extensions (with the exception of Beaver [14], which is not black-box and not concretely efficient) required variants of the LPN assumptions [75, 22, 21, 67].

Precomputability. A nice feature of our two-round setup is the ability for one party to *precompute* all correlations *before even knowing the identity of the other party*. To see this, note that the receiver can precompute all choice bits just using the IPM-wPRF key $\mathbf{z}$ without needing to know the constrained key. Additionally, the receiver can sample a *uniformly random* constrained key $\mathbf{z_1}$ for the ShCPRF and use it to generate ahead-of-time all its ListOT triples (b, i, v). Later, once the identity of the sender is known, the sender can engage with the receiver in a two-round OT protocol to compute the master key $\mathbf{z_0} = \mathbf{z_1} + \Delta \mathbf{z}$ *from* the "constrained key" $\mathbf{z_1}$. Similarly, the sender can alternatively generate all its ListOT instances (L_0, L_1) without needing to know the identity of the receiver by locally sampling Δ and $\mathbf{z_0}$. We provide details on precomputability and more motivation for the notion in Appendix C.1.

2.4 Public-key setup from Ring-LWE

We present a non-interactive distributed setup protocol from RingLWE. To the best of our knowledge, this forms the first distributed setup protocol for PCF for ListOT based on a plausibly post-quantum

assumption. The goal of this protocol is for the sender with input Δ and receiver with input $\mathbf{z}$ to distributively derive keys $\mathbf{z_0}$ (part of the master secret key) and $\mathbf{z_1}$ (the constrained key), which can be viewed as additive shares of $\Delta \cdot \mathbf{z}$ in a ring $\mathcal{R}$.

Parameters. In order to rely on the security of RingLWE, the receiver will "encode" the bits of $\mathbf{z} \in \mathcal{R}^n$ into the coefficients of an element z of a suitable polynomial ring $\mathcal{P}$. The protocol is executed over $\mathcal{P}$ and then, at the end of the protocol, the sender and receiver each "decode" their result back into the ring $\mathcal{R}$ to obtain vectors $\mathbf{z_0}$ and $\mathbf{z_1}$, by parsing each polynomial as a vector of n coefficients (and disregarding any extra coefficients).

Assume that $\mathcal{R} = \mathbb{Z}_t$ is an integer ring, and let $q := n \cdot t \cdot B \cdot 2^{\omega(\log \lambda)}$ (B is some bound on the noise that we compute later). We define $\mathcal{P} := \mathbb{Z}_q[X]/(X^\eta + 1)$, where η is a power of 2 that is larger than n. Let $\chi = \chi(\mathcal{P})$ be a suitable noise distribution over $\mathcal{P}$, such that for $e_0, e_1 \xleftarrow{R} \chi$, it holds that $\|e_0 e_1\|_\infty \leq B/3$, with overwhelming probability.

The protocol proceeds in two phases as follows. During the public-key generation phase, the sender and receiver each broadcast a public key, which is used by the other party in the ShCPRF evaluation key derivation phase.

Step 1: Generating public keys. Fix random $a_0, a_1 \in \mathcal{P}$ as part of the public parameters. To generate public keys, the sender and receiver proceed as follows. These public keys can then be posted to a bulletin board or broadcasted.

Sender

1: Sample secret $s_0 \xleftarrow{R} \chi$.
2: Sample error $e_0 \xleftarrow{R} \chi$.
3: Set $\mathsf{pk}_S = \Delta \cdot a_0 + s_0 a_1 + e_0$.

Receiver

1: Encode $\frac{q}{t} \cdot \mathbf{z}$ as $z \in \mathcal{P}$.
2: Sample secret $s_1 \xleftarrow{R} \chi$.
3: Sample errors $e_1, e_1' \xleftarrow{R} \chi$.
4: Set $\mathsf{pk}_R = (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$.

Step 2: Deriving ShCPRF keys. To derive a master key msk and constrained key csk, respectively, the sender and receiver use the other party's public key to proceed as follows. (Here and throughout, we overload rounding $\lceil \cdot \rfloor_t$ notation to include "rounding" a polynomial coefficient-by-coefficient.)

Sender

1: Compute $z_0 := \lceil \langle \mathsf{pk}_R, (\Delta, s_0) \rangle \rfloor_t$.
2: Decode $z_0 \in \mathcal{P}$ as $\mathbf{z_0} \in \mathcal{R}^n$.
3: Set $\mathsf{msk} := (\mathbf{z_0}, \Delta)$.

Receiver

1: Compute $z_1 := \lceil \mathsf{pk}_S \cdot s_1 \rfloor_t$.
2: Decode $z_1 \in \mathcal{P}$ as $\mathbf{z_1} \in \mathcal{R}^n$.
3: Set $\mathsf{csk} := \mathbf{z_1}$.

Correctness. The inner products computed in the key derivation phase are, in fact, noisy additive shares of $\Delta \cdot z \in \mathcal{P}$, since we have that

$$\begin{aligned}
&\langle \mathsf{pk}_R, (\Delta, s_0) \rangle - (\mathsf{pk}_S \cdot s_1) \\
&= \Delta \cdot z + \Delta \cdot a_0 s_1 + \Delta \cdot e_1 + s_0 a_1 s_1 + s_0 e_1' - \Delta \cdot a_0 s_1 - s_0 a_1 s_1 - e_0 s_1 \\
&= \Delta \cdot z + \underbrace{\Delta \cdot e_1 + s_0 e_1' - e_0 s_1}_{\text{noise}} \approx \Delta \cdot z.
\end{aligned}$$

Note that $\Delta \in \mathbb{Z}_t$ has low norm, so we can bound the magnitude of the noise term $\Delta \cdot e_1 + s_0 e_1' - e_0 s_1$ by B. Hence, by a standard rounding lemma [26, 43], $z_0 - z_1 = \lceil \langle \mathsf{pk}_R, (\Delta, s_0) \rangle \rfloor_t - \lceil \mathsf{pk}_S \cdot s_1 \rfloor_t = \Delta \cdot z \bmod t$. After parsing as vectors over $\mathcal{R}^n$, we have $\mathbf{z_0} - \mathbf{z_1} = \Delta \cdot \mathbf{z}$.

Security. Pseudorandomness of the public keys follows from the RingLWE assumption with short secrets (i.e., *normal form* RingLWE).[11] In the sender public key, the RingLWE sample $s_0 a_1 + e_0$ masks $\Delta \cdot a_0$ and thus the secret key Δ. Similarly, in the receiver public key, the RingLWE sample $s_1 a_0 + e_1$ masks z and thus the secret key $\mathbf{z}$. For a complete description of our protocol, its parameters, and proof of security, we refer to Section 7.5.

[11] Normal form RingLWE is a standard variant of RingLWE where the secret is sampled from the noise distribution instead of uniformly. It is known to be as hard as regular RingLWE and is often used for practical schemes [58, 2, 60, 41].

2.5 Multi-instance security

An immediate application of QuietOT (and public-key OT schemes in general [30, 63]) is for efficient large-scale MPC. At a high level, with QuietOT, parties can, using just the public keys of all other parties, create pair-wise OT channels for the purpose of running a secure computation (e.g., as in the GMW protocol [48]). This application was also described in prior public-key OT constructions [30, 63] but was never formalized. We make the rather subtle observation that existing definitions [30, 63] for public-key OT only require *one-time* security—i.e., privacy for the sender or receiver is not considered when the same public keys are reused with different parties.

To address this gap and properly define public-key OT, we formalize the notion of "multi-instance security" in Section 7. In a nutshell, our definition captures a setting where parties (re)use a *long-term* secret (that depends on the public-key) and an *ephemeral* secret that is generated for each new session. We then prove that our public-key setup satisfies multi-instance security.

3 Preliminaries

3.1 Notation

We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{F}$ denote a finite field, $\mathcal{R}$ denote a finite ring, and $\mathbb{G}$ denote an Abelian group. We denote by $\mathsf{poly}(\cdot)$ any polynomial and by $\mathsf{negl}(\cdot)$ any negligible function.

Sampling and assignment. We let $x \xleftarrow{R} S$ denote a uniformly random sample drawn from S. We let $x \leftarrow \mathcal{A}$ denote assignment from a possibly randomized algorithm $\mathcal{A}$. We let $x := y$ denote initialization of x to the value of y.

Vectors and matrices. We denote a vector $\mathbf{v}$ using bold lowercase letters and a matrix $\mathbf{A}$ using bold uppercase letters. The i-th coordinate of a vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$ (we will also occasionally abuse notation and write $\mathbf{v}[i]$ to lookup a value associated with key i in a key-value list). The i-th bit of a bit-string s is denoted by s_i. For a ring $\mathcal{R}^n$, we define $\Delta \cdot \alpha$ for $\alpha \in \mathcal{R}^n$ as the coordinate-wise scalar product.

Efficiency. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a Turing Machine that runs in probabilistic polynomial time.

Indistinguishability. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

3.2 Cryptographic definitions

Here, we recall the cryptographic definitions that we will use throughout the paper. In Section 3.2.1, we define the notion of an IPM-wPRF. In Section 3.2.2, we cover the definition of Ring-LWE and the basics of modular rounding.

3.2.1 Inner-Product Membership wPRF.

We define the notion of a weak PRF (wPRF)[12] that can be evaluated using the "inner-product membership" formalism introduced by Bui et al. [30].

Definition 1 (Inner-Product Membership wPRF (IPM-wPRF) [30]). *Let λ be the security parameter and $\mathcal{R} = \mathcal{R}$ be a finite ring. Let $S_0 = S_0^{(\lambda)}$ be a (polynomially-sized) subset of $\mathcal{R}_\lambda$, and set $S_1 := \mathcal{R} \setminus S_0$. Then, $f := f_\lambda \colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}$ is an inner-product membership weak PRF (IPM-wPRF) family with respect to the partitioning (S_0, S_1), if it satisfies the following three properties:*

(1) $\mathcal{K}_\lambda = \mathcal{X}_\lambda = \mathcal{R}_\lambda^n$ for some $n = n(\lambda)$,
(2) its evaluation can be expressed as an inner product membership, i.e., for each $\lambda \in \mathbb{N}$, $\mathbf{z} \in \mathcal{K}_\lambda$, $\mathbf{x} \in \mathcal{X}_\lambda$, we have that

$$f_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 0, & \text{if } \langle \mathbf{z}, \mathbf{x} \rangle \in S_0 \\ 1, & \text{otherwise (i.e., } \langle \mathbf{z}, \mathbf{x} \rangle \in S_1\text{),} \end{cases}$$

where $\langle \cdot, \cdot \rangle$ is the standard (simple) inner product on $\mathcal{R}^n$, and
(3) it achieves the standard notion of a secure (weak) PRF [55].

[12] A weak PRF is pseudorandom on *uniformly random* inputs.

3.2.2 Ring Learning with Errors and Rounding.

We recall the standard Ring Learning-with-Errors (RingLWE) assumption of Lyubashevsky, Peikert, and Regev [59] and its normal form.

Definition 2 (The RingLWE assumption [59]). *Let $\eta = \eta(\lambda), q = q(\lambda) \in \mathbb{N}$. Define the polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ and let $\chi = \chi(\lambda)$ be an error distribution over $\mathcal{P}$. The $\mathsf{RingLWE}_{\eta,q,\chi}$ assumption states that for any $t = t(\lambda) \in \mathsf{poly}(\lambda)$, it holds that*

$$(\mathbf{a}, s \cdot \mathbf{a} + \mathbf{e}) \approx_c (\mathbf{a}, \mathbf{u}),$$

where $s \xleftarrow{R} \mathcal{P}, \mathbf{a} \xleftarrow{R} \mathcal{P}^t, \mathbf{e} \xleftarrow{R} \chi^t, \mathbf{u} \xleftarrow{R} \mathcal{P}^t$. The "normal form" $\mathsf{RingLWE}_{\eta,q,\chi}$ assumption states that this holds even when $s \xleftarrow{R} \chi$, and has the same hardness as the original formulation [58, Lemma 2.24].

Modular rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Rounding lemma. We recall the following "rounding lemma" [39, 43, 26]:

Lemma 1 (Rounding of noisy secret shares). *Let (t, q) be two integers such that t divides q. Fix any $z \in \mathbb{Z}_q$ and let (z_0, z_1) be any two random elements of $\mathbb{Z}_q$ subject to $z_0 + z_1 = (q/t) \cdot z + e \bmod q$, where e is such that $q/(t \cdot |e|) \geq \lambda^{\omega(1)}$. Then, with probability at least $1 - (|e| + 1) \cdot t/q \geq 1 - \lambda^{-\omega(1)}$, it holds that $\lfloor z_0 \rceil_t + \lfloor z_1 \rceil_t = z \bmod t$, and the probability is over the random choice of $(z_0, z_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$.*

4 Shiftable CPRFs

In this section, we start by defining the notion of *Shiftable* CPRFs in Section 4.1. Then, in Section 4.2, we construct ShCPRFs for inner-product predicates by adapting the framework of Servan-Schreiber [71].

4.1 Defining Shiftable CPRFs

For simplicity, we restrict the definition to 1-key (rather than multi-key) ShCPRFs and selective security, which is the definition that is satisfied by our construction. The definition overlaps significantly with the definition of (non-shiftable) CPRFs [20, 56, 25] but using the classic PRF-style "real-or-random" security game, where all evaluations are either computed using the master key or using a truly random function.

Remark 1 (On the choice of security game). By using the real-or-random security definition, we manage to get a tight reduction when proving security of our constructions. In contrast, prior work that uses CPRFs to construct PCFs [37, 30] has a polynomial loss in security in their security proofs, proportional to the number of PCF evaluations, which was an artifact of the original CPRF definition.

Remark 2 (Relation to Shift-Hiding Shiftable Functions (SHSF)). SHSF are an extension to CPRFs introduced by Peikert and Shiehian [64]. In a SHSF, the constrained CPRF evaluation computes $F.\mathsf{Eval}(\mathsf{msk}, x) + f(x)$, for a hidden "shift" function f embedded into the constrained key. In contrast, our notion of Shiftable CPRFs only allows the master key holder to shift the *constraint* when evaluating the CPRF (using the master key) and does not affect the constrained key.

Definition 3 (Shiftable Constrained Pseudorandom Functions). *Let $\lambda \in \mathbb{N}$ be a security parameter. A Shiftable Constrained Pseudorandom Function (ShCPRF) with domain $\mathcal{X} = \mathcal{X}_\lambda$, range $\mathcal{Y}$, and a finite set of shifts $\mathcal{S}$ that supports constraints represented by the class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where $\mathcal{C}_\lambda \colon \mathcal{X} \times \mathcal{S} \to \{0, 1\}$, consists of the following four algorithms. We highlight the parts that are specific to shiftable CPRFs.*

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{msk}$. *Takes as input a security parameter λ. Outputs a master secret key* msk.
- $\mathsf{Eval}(\mathsf{msk}, x, \alpha) \to y$. *Takes as input the master secret key* msk, *an input $x \in \mathcal{X}$, and a shift $\alpha \in \mathcal{S}$. Outputs $y \in \mathcal{Y}$.*
- $\mathsf{Constrain}(\mathsf{msk}, C) \to \mathsf{csk}$. *Takes as input the master secret key* msk *and a constraint circuit $C \in \mathcal{C}$. Outputs a constrained key* csk.

– $\mathsf{CEval}(\mathsf{csk}, x) \to y$. *Takes as input the constrained key* csk *and an input* $x \in \mathcal{X}$. *Outputs* $y \in \mathcal{Y}$.

We let any public parameters PP *be an implicit input to all algorithms. A ShCPRF must satisfy the following correctness, security, and pseudorandomness properties. We let* $\widetilde{\mathcal{F}} = \widetilde{\mathcal{F}}_\lambda$ *denote the set of all functions from* $\mathcal{X} \times \mathcal{S}$ *to* $\mathcal{Y}$.

Correctness. *For all security parameters* λ, *all constraints* $C \in \mathcal{C}$, *and all inputs* $x \in \mathcal{X}$, *there exists an efficiently computable* $\alpha \in \mathcal{S}$ *such that* $C(x, \alpha) = 0$ *(authorized), and for all* $\alpha \in \mathcal{S}$ *where* $C(x, \alpha) = 0$ *it holds that:*

$$\Pr\left[\begin{matrix}\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)\end{matrix} \;:\; \mathsf{Eval}(\mathsf{msk}, x, \alpha) = \mathsf{CEval}(\mathsf{csk}, x)\right] = 1 - \mathsf{negl}(\lambda),$$

where the probability space is over the randomness used in KeyGen *and* $\mathsf{Constrain}$.

(1-key, selective) Security. *A ShCPRF is (1-key, selectively)-secure if for all efficient adversaries* $\mathcal{A}$, *the advantage of* $\mathcal{A}$ *in the following security experiment* $\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},b}(\lambda)$ *is negligible in* λ. *Here,* b *denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ, *the challenger*
   – *runs* $\mathcal{A}(1^\lambda)$ *who outputs a constraint* $C \in \mathcal{C}$,
   – *computes* $\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$,
   – *samples a uniformly random function* $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, *and*
   – *sends* csk *to* $\mathcal{A}$.
2. ***Evaluation queries:*** $\mathcal{A}$ *adaptively sends arbitrary inputs* $x \in \mathcal{X}$ *and shifts* $\alpha \in \mathcal{S}$ *to the challenger. For each pair* (x, α), *if* $C(x, \alpha) = 0$, *then the challenger returns* $\perp$. *Otherwise, the challenger proceeds as follows:*
   – *If* $b = 0$, *it computes* $y \leftarrow \mathsf{Eval}(\mathsf{msk}, x, \alpha)$ *and returns* y.
   – *If* $b = 1$, *it computes* $y \leftarrow R(x, \alpha)$ *and returns* y.
3. ***Guess:*** $\mathcal{A}$ *outputs its guess* b', *which is the output of the experiment.*

$\mathcal{A}$ *wins if* $b' = b$, *and its advantage* $\mathsf{Adv}^{\mathsf{shcprf}}_{\mathcal{A}}(\lambda)$ *is defined as*

$$\mathsf{Adv}^{\mathsf{shcprf}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ *and* KeyGen.

Pseudorandomness. *A ShCPRF is said to be pseudorandom if for all efficient adversaries* $\mathcal{A}$, *it holds that*

$$\left|\Pr_{\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)}[\mathcal{A}^{\mathsf{Eval}(\mathsf{msk}, \cdot, \cdot)}(1^\lambda) = 1] - \Pr_{R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda}[\mathcal{A}^{R(\cdot,\cdot)}(1^\lambda) = 1]\right| = 1 - \mathsf{negl}(\lambda).$$

Remark 3 (The requirement for the pseudorandomness property). We note that while the pseudorandomness property is implicit in standard CPRF definition (which has a polynomial loss in security) [20], in the real-or-random style definition for CPRFs (as in Definition 3), this property is not immediately satisfied, since the challenger outputs $\perp$ when given an unconstrained query. Hence, we require the separate *pseudorandomness* property to capture the requirement that the function being constrained is indeed a standard PRF.

4.2 Constructing Shiftable CPRFs

In this section, we adapt the framework of Servan-Schreiber [71] constructing CPRFs for inner-product predicates from RKA-secure PRFs. We make the observation that the construction can be easily adapted to fit the shiftable CPRF definition (Definition 3). In the process, we additionally generalize the construction of Servan-Schreiber to work over a small ring as opposed to a large field, which makes it integrate better with an IPM-wPRF as the predicate.

The CPRF framework of Servan-Schreiber in a nutshell. The CPRF of [71] is parameterized by a security parameter λ, finite field $\mathbb{F}$ of order at least 2^λ, and a vector length parameter $n \geq 1$. The master secret key msk consists of a random vector $\mathbf{z_0} \in \mathbb{F}^n$. The constrained key csk for a constraint $\mathbf{z} \in \mathbb{F}^n$ is then defined as $\mathbf{z_1} := \mathbf{z_0} - \Delta\mathbf{z}$, with $\Delta \in \mathbb{F} \setminus \{0\}$ a random non-zero scalar. The main insight behind the framework of [71] is that for an input $\mathbf{x} \in \mathbb{F}^n$, when $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e., when the constraint is satisfied), then the inner product $\langle \mathbf{z_0}, \mathbf{x} \rangle$ is equal to $\langle \mathbf{z_1}, \mathbf{x} \rangle$. This fact can be used to derive *identical* PRF keys k and k' under both msk and csk:

$$k = \langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle - \cancel{\Delta \langle \mathbf{z}, \mathbf{x} \rangle} = \langle \mathbf{z_1}, \mathbf{x} \rangle = k'.$$

In contrast, when $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the $\Delta \langle \mathbf{z}, \mathbf{x} \rangle$-term makes $k \neq k'$. Moreover, because Δ is uniformly random over $\mathbb{F} \setminus \{0\}$ (where $\mathbb{F}$ has order at least 2^λ), $\mathbf{z_1}$ cannot be used to recover $\mathbf{z_0}$, even with knowledge of the constraint $\mathbf{z}$. The evaluation of the CPRF is then defined as $F_{k_0+k}(\mathbf{x})$ (resp. $F_{k_0+k'}(\mathbf{x})$ for the constrained evaluation), where k_0 is a "zero" PRF key used to handle the case where $\mathbf{x} = 0^n$. One caveat, however, is that the derived PRF keys are *highly correlated*, which necessitates choosing F to be a suitable RKA-secure PRF. The work of Servan-Schreiber shows that when the PRF F is RKA-secure for affine key-derivation functions (Definition 12), then the CPRF instantiated with the PRF F is secure. (We note that a random oracle $H \colon \mathbb{F} \times \mathbb{F}^n \to \{0,1\}^*$ is RKA-secure PRF for all non-trivial key-derivation functions.)

Adding shiftability. We make the simple observation that if we make the master secret key msk also contain Δ, then the we can easily turn the above framework into a *shiftable* CPRF as follows. Specifically, when $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the constrained key computes $k' = \langle \mathbf{z_0}, \mathbf{x} \rangle - \Delta \langle \mathbf{z}, \mathbf{x} \rangle$. The master key holder, with knowledge of Δ, can compute $k = \langle \mathbf{z_0}, \mathbf{x} \rangle - \Delta \cdot \alpha = k'$, where $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$. This is enough to satisfy the correctness property of Definition 3 (Shiftable CPRFs). In particular, here the constraint predicate $C(\mathbf{x}, \alpha)$ is 0 if $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$ and 1 otherwise.

Moving to the ring setting. We require instantiating the ShCPRF with a *small* ring $\mathcal{R}$ (e.g., $\mathcal{R} = \mathbb{Z}_6$) for efficiency purposes. However, to ensure each derived key is still at least λ-bits, we must extend the small ring to a sufficiently large ring $\mathcal{R}'$. As such, we replace the large field $\mathbb{F}$ with a large $\mathcal{R}' = \mathcal{R}^m$ where $m \geq \lambda$ to guarantee λ bits of security (we prove the security of this modification in Lemma 2 of Appendix A). Doing so, however, makes the vectors $\mathbf{z_0}$ and $\mathbf{z_1}$ (now sampled in $(\mathcal{R}')^n$) better denoted as *matrices* from $\mathcal{R}^{m \times n}$. While this induces notational changes, the CPRF construction itself remains almost identical to the one of Servan-Schreiber. In particular, for a constraint $\mathbf{z} \in \mathcal{R}^n$, we now have CPRF keys $\mathbf{k}, \mathbf{k}' \in \mathcal{R}^m$ (as opposed to $k, k' \in \mathbb{F}$ above) derived for an input $\mathbf{x} \in \mathcal{R}^n$ as $\mathbf{k} = \mathbf{Z_0 x} = \mathbf{Z_1 x} + (\Delta \mathbf{z}^\top)\mathbf{x}$ which is equal to $\mathbf{k}' = \mathbf{Z_1 x}$, when the constraint $\langle \mathbf{z}, \mathbf{x} \rangle = 0 \in \mathcal{R}$. We present our ring-based ShCPRF framework in Figure 1 and show security in Section 4.3 and Appendix D.1.

ShCPRF for Inner-Product Predicates

Public Parameters. Security parameter λ, finite ring $\mathcal{R}$ of order ℓ, integers m such that $m \geq \lambda$, vector length $n \geq 1$, and an RKA-secure PRF family $F \colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ for affine key-derivation functions.

$\mathsf{ShCPRF.KeyGen}(1^\lambda)$:
1 : $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m,\ \Delta \xleftarrow{R} \mathcal{R}^m \setminus \{0\}$
2 : $\mathbf{Z_0} \xleftarrow{R} \mathcal{R}^{m \times n}$
3 : $\mathsf{msk} := (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$

$\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$:
1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$
2 : $\mathbf{k} \leftarrow \mathbf{k_0} + \mathbf{Z_0 x} - \Delta \cdot \alpha$
3 : **return** $F_{\mathbf{k}}(\mathbf{x})$

$\mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$:
1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$
2 : $\mathbf{Z_1} \leftarrow \mathbf{Z_0} - \Delta \mathbf{z}^\top$
3 : **return** $\mathsf{csk} := (\mathbf{k_0}, \mathbf{Z_1})$

$\mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$:
1 : **parse** $\mathsf{csk} := (\mathbf{k_0}, \mathbf{Z_1})$
2 : $\mathbf{k} \leftarrow \mathbf{k_0} + \mathbf{Z_1 x}$
3 : **return** $F_{\mathbf{k}}(\mathbf{x})$

Fig. 1: ShCPRF framework for inner-product predicates based on RKA-secure PRFs.

To state more exactly the special type of Shiftable CPRF we obtain, we have the following definition.

Definition 4 (Shiftable CPRFs for Inner-Product Predicates). *Let $\mathcal{R} = \mathcal{R}_\lambda$ be a finite ring. Let* ShCPRF *be a Shiftable CPRF with domain $\mathcal{X} = \mathcal{R}^n$ for an $n = n(\lambda)$, range $\mathcal{R}$, and finite set of shifts $\mathcal{S} = \mathcal{R}$ that supports constraints represented by a class of circuits $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, such that $\mathcal{C}_\lambda = \{C_{\mathbf{z}} \colon \mathbf{z} \in \mathcal{R}^n\}$, where the $C_{\mathbf{z}} \colon \mathcal{X} \times \mathcal{S} \to \{0,1\}$ are given via*

$$(\mathbf{x}, \alpha) \mapsto \begin{cases} 0, & \text{if } \langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0, \\ 1, & \text{otherwise.} \end{cases}$$

Then we identify the constraint circuit $C_{\mathbf{z}}$ with $\mathbf{z}$, i.e. we just write that the constraint is a vector $\mathbf{z}$, and call ShCPRF *a (ring-based)* Shiftable CPRF for inner-product predicates *(with domain $\mathcal{R}^n$).*

4.3 Security analysis

Here, we analyze the security of the ShCPRF framework using the proof template of Servan-Schreiber [71]. The proof follows the proof of [71, Theorem 2], however, we adapt it in several key locations to handle the shiftability property and operations in the ring $\mathcal{R}$.

Theorem 1. *If $\mathcal{F}$ is a family of RKA-secure pseudorandom functions with respect to affine related-key derivation functions Φ_{aff}, as defined in Definition 12, then Figure 1 instantiated with $\mathcal{F}$ is a (1-key, selectively-secure) ShCPRF for inner-product constraint predicates.*

Proof. Deferred to Appendix D.1. ■

5 PCFs for ListOT: Framework

In this section, we formalize our PCF for ListOT framework using the Shiftable CPRF framework from Section 4. As mentioned in Section 2, ListOT does *not* fulfill the definition of a "correlation" as defined by Boyle et al. [24]. Therefore, we cannot use existing definitions of a Pseudorandom *Correlation* Function (PCF), since the correlation is only partially defined. In particular, the problem is that the output of the receiver in ListOT has an additional lookup key i that depends directly on the wPRF key used to compute the pseudorandom bit b, which cannot be efficiently sampled given just the output of the sender. We sidestep these issues by adapting the standard definition of a PCF [23] to work with the "partial correlation" that is ListOT. In Section 5.1, we define the notion of a PCF for ListOT. Then, in Section 5.2, we describe our general framework for constructing a PCF for ListOT. Finally, in Section 5.3, we explain how a PCF for ListOT is used to instantiate QuietOT.

5.1 Defining PCFs for ListOT

Here, we give a formal definition of (weak) PCF for ListOT. For convenience, we use of the following *distribution of lists* notation.

Definition 5 (Distribution of lists). *Let λ be a security parameter, $\mathcal{Y}$ be a finite set, and I be an arbitrary finite index set. We denote by $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(I)$ the distribution that outputs a list $(v_i)_{i\in I}$, where each $v_i \xleftarrow{R} \mathcal{Y}$ is independently sampled at random.*

Definition 6 (Pseudorandom Correlation Function for ListOT). *Let λ be a security parameter and $\lambda \le n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Pseudorandom Correlation Function (PCF) for ListOT with domain $\mathcal{X} = \mathcal{X}_\lambda$ is defined by a pair of algorithms* (PCF.KeyGen, PCF.Eval) *with the following functionality:*

- $\mathsf{PCF.KeyGen}(1^\lambda) \to (K_S, K_R)$. *Takes as input the security parameter λ. Outputs a pair of keys (K_S, K_R).*
- $\mathsf{PCF.Eval}(\sigma, K_\sigma, x) \to y_\sigma$. *Takes as input $\sigma \in \{S, R\}$, a key K_σ, and input $x \in \mathcal{X}$. Outputs a string $y_\sigma \in \{0,1\}^*$, where*
  - *if $\sigma = S$ then $y_\sigma = (L_0, L_1)$ for two lists L_0, L_1, and*

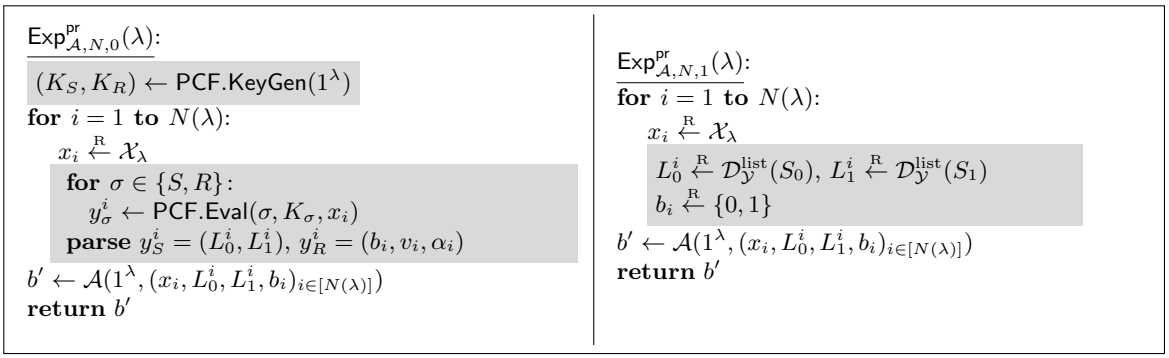

Fig. 2: (Partially) Pseudorandom outputs of a PCF for ListOT. The distribution $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(\cdot)$ is defined in Definition 5.

- *if $\sigma = R$ then $y_\sigma = (b, v, \alpha)$ for a bit $b \in \{0,1\}$, a list entry $v \in L_0 \cup L_1$, and a lookup key α.*

We will use $\mathsf{PCF.EvalS}(K_S, x)$ *and* $\mathsf{PCF.EvalR}(K_R, x)$ *as shorthand for the* Eval *algorithm used by the sender and receiver, respectively. We leave any public parameters* PP *as an implicit input to all algorithms.*

A $\mathsf{PCF} = (\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ *is a* (weak) PCF for ListOT *with domain $\mathcal{X} = \mathcal{X}_\lambda$, if the following* correctness, sender security, *and* receiver security *properties hold. In each case, the adversary is given access to $N(\lambda) \in \mathsf{poly}(\lambda)$ samples.*

– ***Pseudorandomness.*** *For all efficient adversaries $\mathcal{A}$, and all $N \in \mathsf{poly}(\lambda)$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 2.

– ***Correctness.*** *Moreover, we want that for any $\lambda \in \mathbb{N}$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{l|l} (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda), & \\ x \xleftarrow{R} \mathcal{X}_\lambda, & \\ (L_0, L_1) \leftarrow \mathsf{PCF.EvalS}(K_S, x), & v \neq L_b[\alpha] \\ (b, v, \alpha) \leftarrow \mathsf{PCF.EvalR}(K_R, x) & \end{array}\right],$$

i.e., that the (relevant) entry v is at position α of list L_b with a probability that is overwhelming in λ.

– ***Sender Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{Ssec}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 3.

– ***Receiver Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{Rsec}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 4.

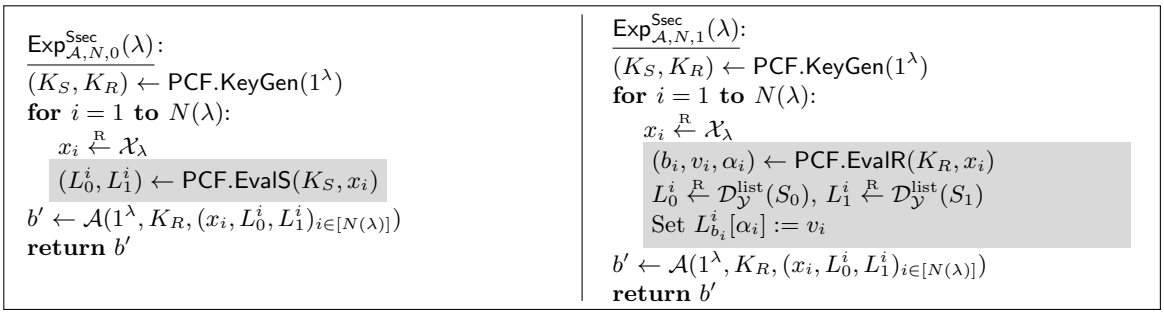

Fig. 3: Sender security game of a PCF for ListOT. The distribution $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(\cdot)$ is defined in Definition 5.

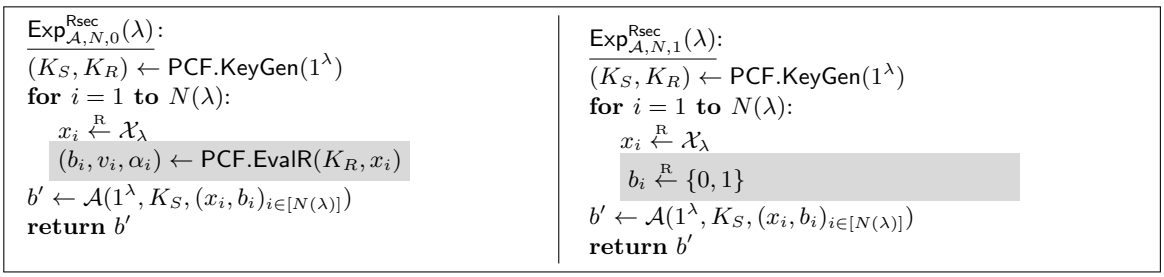

Fig. 4: Receiver security game of a PCF for ListOT.

5.2 Framework: PCF for ListOT from IPM-wPRFs

In Figure 5, we describe the framework for constructing a PCF for ListOT by combining a Shiftable CPRF with any IPM-wPRF.

Theorem 2. *Let $n = n(\lambda) \in \mathsf{poly}(\lambda)$ and $\mathcal{R}$ be a finite ring of order q, with extension parameter $m \geq \lambda$. Let $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ be a shiftable CPRF for inner-product predicates with domain $\mathcal{R}^n$, and $f\colon \mathcal{R}^n \times \mathcal{R}^n \to \{0,1\}$ a weak PRF family for inner-product membership with partitioning $S_0 \cup S_1 = \mathcal{R}$. Then, $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ from Figure 5 is a PCF for ListOT.*

Proof. Deferred to Appendix D.2. ■

5.3 Realizing QuietOT from a PCF for ListOT

To generate random OT correlations, the sender and receiver use the PCF for ListOT to generate pseudorandom ListOT instances. We use the template of Beaver [15] for converting a random ListOT instance into a chosen-bit OT protocol. We describe this transformation in Figure 6.

Proposition 1. *Let $\mathsf{PCF} = (\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ be a PCF for ListOT. Then the protocol given in Figure 6 securely realizes the OT functionality.*

Proof. The OT functionality is defined in Appendix A.1. By the pseudorandomness property of the PCF for ListOT we have that L_0 and L_1 are pseudorandom lists and b' is a pseudorandom bit if x (input to the PCF) is uniformly random. For an arbitrary choice bit $b \in \{0,1\}$ we consider the two possible cases to prove correctness.

- Case 1: $b = 0$. In this case, $c = b'$ and so $L'_0 = L_{b'} \oplus m_0$ and $L'_1 = L_{1-b'} \oplus m_1$. It then follows that the receiver outputs $(L'_0[\alpha] \oplus m_0) \oplus v$, which equals m_0 by the correctness property of the PCF.
- Case 2: $b = 1$. In this case, $c = 1 - b'$ and so $L'_0 = L_{1-b'} \oplus m_0$ and $L'_1 = L_{b'} \oplus m_1$. It then follows that the receiver outputs $m_1 = (L'_1[\alpha] \oplus m_1) \oplus v$, since we have $v = L_{b'}[\alpha] = L'_1[\alpha]$ by the correctness property of the PCF.

Note that in both cases, the equality holds with overwhelming probability, because correctness of the PCF holds with overwhelming probability (Definition 6).

Sender security follows directly from the sender security of the PCF which guarantees that (1) the receiver only obtains $L_{b'}[\alpha]$ and (2) all other values in both lists are pseudorandom from the viewpoint of the receiver and therefore guarantees that the receiver only obtains m_b.

PCF for ListOT

Public Parameters (PP).

- Key length $n = n(\lambda) \in \mathsf{poly}(\lambda)$.
- Finite ring $\mathcal{R}$ of order q, with extension parameter $m \geq \lambda$.
- IPM-wPRF family $f : \mathcal{R}^n \times \mathcal{R}^n \to \{0,1\}$ with partitioning $S_0 \cup S_1 = \mathcal{R}$.
- An injective input mapping $\mathsf{map}\colon \mathcal{X}_\lambda \to \mathcal{R}^n$.
- A ShCPRF for inner-product predicates $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ with domain $\mathcal{R}^n$

$\mathsf{PCF.KeyGen}(1^\lambda)$**.**

1: $\mathsf{msk} \leftarrow \mathsf{ShCPRF.KeyGen}(1^\lambda)$
2: $\mathbf{z} \xleftarrow{\text{R}} \mathcal{R}^n$. ▷ IPM-wPRF key
▷ Note that $\mathbf{z}$ can also be sampled from a non-uniform distribution over $\mathcal{R}^n$.
3: $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$
4: $K_S := \mathsf{msk}$, $K_R := (\mathsf{csk}, \mathbf{z})$.
5: **return** (K_S, K_R)

$\mathsf{PCF.EvalS}(K_S, x)$**.**

1: **parse** $K_S = \mathsf{msk}$.
2: $\mathbf{x} \leftarrow \mathsf{map}(x)$.
3: **foreach** $b \in \{0,1\}$ **and** $\alpha \in S_b$:
    1: $y \leftarrow \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$.
    2: $L_b[\alpha] = y$.
4: **return** (L_0, L_1).

$\mathsf{PCF.EvalR}(K_R, x)$**.**

1: **parse** $K_R = (\mathsf{csk}, \mathbf{z})$.
2: $\mathbf{x} \leftarrow \mathsf{map}(x)$.
3: $v \leftarrow \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$
4: $b \leftarrow f_{\mathbf{z}}(\mathbf{x})$, $\alpha \leftarrow \langle \mathbf{z}, \mathbf{x} \rangle \in \mathcal{R}$.
▷ Note that $v = L_b[\alpha]$ in the sender-computed list.
5: **return** (b, v, α).

Fig. 5: Framework for a PCF for ListOT from any ShCPRF and IPM-wPRF.

Receiver security follows from the fact that b' is a pseudorandom bit (by receiver security of the PCF) and therefore a pseudorandom mask for the receiver's choice bit b. Therefore, the sender learns nothing. ■

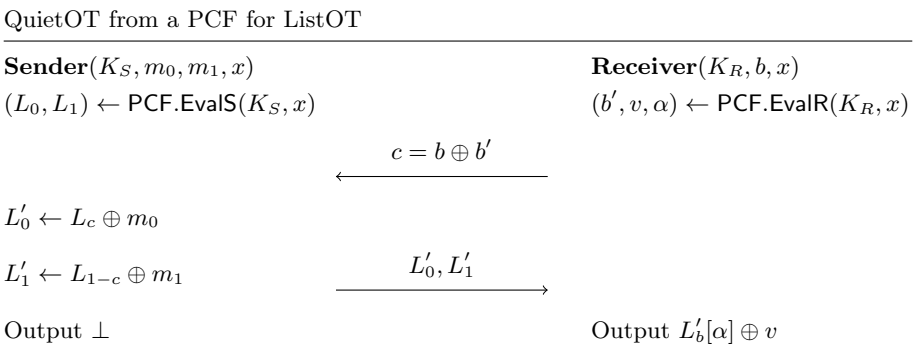

Fig. 6: QuietOT using a (weak) PCF for ListOT. Note that x (input to the PCF known by both the sender and receiver) is uniformly random, as specified in Definition 6.

6 PCFs for ListOT: Instantiations

In this section, we instantiate the framework from Section 5 using either the BIPSW or GAR IPM-wPRF candidate, coupled with the "RKA-PRF" $F_k(x) := H(k, x)$ for a hash function H modeled as a random oracle. For the sake of completeness, we prove in Appendix A.3 that F_k is indeed an RKA-PRF for all affine relations $x \mapsto \alpha x + \beta$ with $\alpha \in \mathcal{R}^*$ and $\beta \in \mathcal{R}^m$, as long as $\min_\alpha(|\alpha \cdot \mathcal{R}^m|) \geq 2^\lambda$. Looking ahead, both our instantiations will satisfy $\min_\alpha(|\alpha \cdot \mathcal{R}^m|) = 2^m$, and we will therefore set $m = \lambda$.

These two instantiations result in our concretely efficient constructions (see Section 8). We also describe other instantiations from different assumptions (in particular, replacing the random oracle using an RKA-secure PRF), which have interesting theoretical implications but do not currently result in concretely efficient constructions.

6.1 BIPSW IPM-wPRF instantiation

Our main instantiation is based on the BIPSW wPRF candidate which can be easily viewed as an IPM-wPRF. For a key $\mathbf{z} \in \mathbb{Z}_6^n$ with $n = n(\lambda) \in \mathsf{poly}(\lambda)$, and $\mathbf{x} \in \mathbb{Z}_6^n$, the BIPSW wPRF is defined as: $f_{\mathbf{z}}(\mathbf{x}) = \lfloor \langle \mathbf{z}, \mathbf{x} \rangle \bmod 6 \rceil_2$, where $\lfloor \alpha \rceil_2 = 0$ for all $\alpha \in \{0, 1, 2\}$ and $\lfloor \alpha \rceil_2 = 1$ for all $\alpha \in \{3, 4, 5\}$. For a partitioning of $\mathbb{Z}_6$ consisting of $S_0 := \{0, 1, 2\}$ and $S_1 := \{3, 4, 5\}$, we get that $\langle \mathbf{z}, \mathbf{x} \rangle \bmod 6 \in S_b \iff f_{\mathbf{z}}(\mathbf{x}) = b$.

We instantiate the framework using the ring $\mathcal{R} = \mathbb{Z}_6$ and set $m \geq \lambda$ (see Lemma 2). We interpret $\{0, 1\}$ as elements of $\mathbb{Z}_6$ in the natural way (mapping 0 to the additive identity and 1 to the multiplicative identity of $\mathbb{Z}_6$) and define map to be the canonical embedding from $\{0, 1\}^n$ to $\mathbb{Z}_6^n$. The full construction is presented in Figure 7 and closely follows the general framework from Figure 5 with the exception that we use the specific ShCPRF construction of Figure 1, and fix the RKA-secure PRF to be the random oracle H, and additionally explicitly work over the ring $\mathbb{Z}_6$.

6.2 GAR IPM-wPRF instantiation

Unlike for the BIPSW wPRF, converting the GAR wPRF into an IPM-wPRF is more challenging. For completeness, we describe how Bui et al. [30] express the evaluation function as an IPM and discuss concrete parameters that we use for our instantiation, which differ from the parameters used to instantiate BCMPR.

PCF for ListOT from the BIPSW IPM-wPRF

Parameters.

- Integers $n = n(\lambda) \in \mathsf{poly}(\lambda)$, $m \geq \lambda$ and domain $\mathcal{X}_\lambda = \{0,1\}^n$.
- BIPSW IPM-wPRF family $f : \mathbb{Z}_6^n \times \mathbb{Z}_6^n \to \{0,1\}$ with partitioning:
$$S_0 := \{\alpha \in \mathbb{Z}_6 \mid \alpha < 3\} \text{ and } S_1 := \mathbb{Z}_6 \setminus S_0.$$
- Input mapping $\mathsf{map}\colon \mathcal{X}_\lambda \to \mathcal{R}^n$ is the canonical embedding of $\{0,1\}^n$ in $\mathbb{Z}_6^n$.
- The ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ from Figure 1, where the RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ is instatiated by a random oracle H, cf. Appendix A.3.

Construction.
$(\mathsf{PCF.KeyGen}, \mathsf{PCF.EvalR}, \mathsf{PCF.EvalS})$ are identical to Figure 5 using $\mathcal{R} = \mathbb{Z}_6$.

Fig. 7: PCF for ListOT from the BIPSW IPM-wPRF

The GAR construction. In a nutshell, the GAR construction (when instantiated with the XOR-MAJ predicate [5, 36]) has a key $K \in \{0,1\}^n$ and takes as input a string x that is parsed as a tuple of disjoint *sets* $(X_{\mathsf{xor}}, X_{\mathsf{maj}}) \subset [n]^2$ such that $|X_{\mathsf{xor}}| = k$ and $|X_{\mathsf{maj}}| = \ell$, for integers $k = k(\lambda), \ell = \ell(\lambda)$. The evaluation of f_K is then computed as: $(\bigoplus_{i \in X_{\mathsf{xor}}} K[i]) \oplus \mathsf{MAJ}((K[j])_{j \in X_{\mathsf{maj}}})$, where MAJ outputs the majority bit.

The GAR construction as an IPM-wPRF. Converting this evaluation into an inner-product membership can be done as follows. View the evaluation as two separate components: an XOR component and a MAJ component. For each index $i \in X_{\mathsf{xor}}$, let $\mathbf{e}_i$ be the one-hot vector of length n with 1 in its i-th coordinate. First, interpret K as $\mathbf{z}_{\mathsf{xor}} \in \mathbb{Z}_2^n$ and as $\mathbf{z}_{\mathsf{maj}} \in \mathbb{Z}_\ell^n$ by mapping 0 to $0 \in \mathbb{Z}_2$ (resp. $\mathbb{Z}_\ell$) and 1 to $1 \in \mathbb{Z}_2$ (resp. $\mathbb{Z}_\ell$)). Then, compute $v_{\mathsf{xor}} = \sum_{i \in X_{\mathsf{xor}}} \langle \mathbf{z}_{\mathsf{xor}}, \mathbf{e}_i \rangle$. Similarly, for each index $j \in X_{\mathsf{maj}}$, let $\mathbf{e}_j$ be the corresponding one-hot vector. Then, compute $v_{\mathsf{maj}} = \sum_{j \in X_{\mathsf{maj}}} \langle \mathbf{z}_{\mathsf{maj}}, \mathbf{e}_j \rangle$. Observe that $v_{\mathsf{maj}} \geq \lceil \frac{\ell}{2} \rceil \iff \mathsf{MAJ}((\mathbf{z}_{\mathsf{maj}}[j])_{j \in X_{\mathsf{maj}}}) = 1$. We define $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$, which intuitively allows for computing the "XOR" and "MAJ" components in separate subrings. Therefore, we can view f as an IPM-wPRF with partition:

$$S_0 = \left\{(u,v) \in \mathcal{R} \mid (u = 0 \wedge v > \lfloor \tfrac{\ell}{2} \rfloor) \vee (u = 1 \wedge v \leq \lfloor \tfrac{\ell}{2} \rfloor)\right\} \text{ and } S_1 = \mathcal{R} \setminus S_0.$$

Parameters. We follow the parameter selection process of Bui et al. [30], which builds upon the state-of-the-art cryptanalysis of Goldreich's PRG from [5, 36, 74, 73]. To achieve λ bits of security with a key of length $n = \lambda^\delta$ and a bound n^{1+e} on the number of queries, the analysis of Bui et al. [30] suggests to use the XOR-MAJ predicate with $\ell_1 = 2 \cdot e + 1$ terms in the XOR, and $\ell_2 = (2\delta/(\delta - 1)) \cdot e + 1$ terms in the MAJ. Concretely, we set $e = 2$ to get a stretch n^3 (looking ahead, we will choose $n = 2^{11}$, hence this corresponds to generating up to 2^{33} OTs) and $\delta = 7/5$ (hence $\delta/(\delta - 1) = 7/2$). This implies that we can set $\ell_1 = 5$ and $\ell_2 = 15$. With these parameters, we must set $\ell = \ell_2 + 1 = 16$ to ensure no wraparound when computing the MAJ predicate on the sum modulo ℓ of the ℓ_2 entries in the corresponding subset. While this analysis indicates that a seed size of $n \geq 128^\delta = 892$ suffices, we set $n = 2048$ which allows us to more efficiently parse uniformly random inputs x into the index sets X_{xor} and X_{maj}, and generate a larger number $\lambda^\delta = 2^{33}$ of oblivious transfers. This results in an extremely conservative parameter set: The estimated bit security of this parameter set, using the state-of-the-art cryptanalysis [5, 36, 74, 73], is 2^{232}.

6.3 Other instantiations

While we focus on the BIPSW and GAR IPM-wPRF constructions when instantiating our framework, several other instantiations are possible. First, we can instantiate the framework using a different IPM-wPRF candidate. While BIPSW and GAR appear to be the most efficient candidates to fit the IPM-wPRF template, future wPRF candidates or improved parameters for the LWR wPRF resulting from tighter reductions for the LWR problem, could lead to new instantiations. For example, with the VDLPN wPRF candidate [23] (which can be cast as an IPM-wPRF [30]) and whose concrete security is beginning to be analyzed [35], we could potentially have an additional practical instantiation.

PCF for ListOT from the GAR IPM-wPRF

Parameters.

- Integers $n = n(\lambda) \in \mathsf{poly}(\lambda)$ and $m \geq \lambda$ and $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$.
- GAR IPM-wPRF family $f : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ with parameters $w_0, w_1, w = w(w_0, w_1)$, where $w_0, w_1, w \in \mathbb{N}$, and partitioning:
$$S_0 = \left\{(u,v) \in \mathcal{R} \mid (u = 0 \wedge v > \left\lfloor \tfrac{\ell}{2} \right\rfloor) \vee (u = 1 \wedge v \leq \left\lfloor \tfrac{\ell}{2} \right\rfloor)\right\} \text{ and } S_1 = \mathcal{R} \setminus S_0.$$
- Input mapping $\mathsf{map}\colon \{0,1\}^{\mathsf{poly}(n)} \to \mathcal{R}^n$ where each uniformly random input $x \in \{0,1\}^{\mathsf{poly}(n)}$ is interpreted as a tuple $(x_{\mathsf{xor}}, x_{\mathsf{maj}}) \in \{0,1\}^n \times \{0,1\}^n$ subject to $\mathcal{HW}(x_{\mathsf{xor}}) = w_0$ and $\mathcal{HW}(x_{\mathsf{maj}}) = w_1$, where $\mathcal{HW}(\cdot)$ denotes the Hamming weight of the input. $(x_{\mathsf{xor}}, x_{\mathsf{maj}})$ is interpreted as $(\mathbf{x}_{\mathsf{xor}}, \mathbf{x}_{\mathsf{maj}}) \in \mathcal{R}^n$ in the natural way (by interpreting 0 and 1 as elements of $\mathcal{R}$).
- The ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ from Figure 1, where the RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ is instatiated by a random oracle H, cf. Appendix A.3.

Construction.

$(\mathsf{PCF.KeyGen}, \mathsf{PCF.EvalR}, \mathsf{PCF.EvalS})$ are as described in Figure 5 using the ring $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$ and input mapping map defined above.

Fig. 8: PCF for ListOT from the GAR IPM-wPRF

Additionally, our framework is not restricted to the random oracle model (albeit, assuming a random oracle can lead to the most practical instantiations, as is the case for other OT extension protocols). As with prior OT extension protocols, we can replace the random oracle with a suitable correlation-robust hash function [53]. However, we can even go one step further. Note that because the security relies on the security of the ShCPRF, and the CPRF construction of Servan-Schreiber [71] relies only any suitable RKA-secure PRF (a property inherited by our construction of shiftable CPRFs), we can instantiate it using other assumptions such as DDH, DCR, or VDLPN. We discuss these (currently purely theoretical) instantiations in Appendix C.3.

7 Public-Key Setup

We define the notion of a public-key setup for our PCF for ListOT. These definitions provide the necessary foundations to apply public-key OT to an MPC setting (e.g., where parties may engage in pair-wise OT extensions). While the application of public-key OT to MPC was mentioned in several prior works [30, 63], they did not provide a formal treatment of this application. We find that the standard definition of public-key OT only guarantees "one-instance" security, and does not address the many subtleties that arise when the existing constructions are applied to a multi-party setting where an adversary can corrupt multiple parties. Unfortunately, this makes existing public-key OT constructions potentially insecure if used in such contexts. Here, we lay down the foundations necessary for providing "ℓ-instance"-secure public-key OT, and prove that our main framework satisfies this stronger definition.

7.1 ℓ-instance updatability of shiftable CPRFs

Here, we upgrade the definition from Section 5 to provide a notion of updatability, allowing two parties to generate an ShCPRF key from "partial" keys.

Definition 7. *Let* $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ *be a shiftable CPRF with domain* $\mathcal{X} = \mathcal{X}_\lambda$ *and range* $\mathcal{Y}$ *that supports constraints represented by the class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *(recall Definition 3). We say the ShCPRF is* updatable*, if the key generation algorithm* KeyGen *outputs a master secret key* msk *of the form* $(\mathsf{ltsk}, \mathsf{esk})$*, where* ltsk *is a* long-term secret key *part,* esk *is an* ephemeral secret key *part, and there exists an additional efficient algorithm:*

- $\mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*, C) \to \mathsf{esk}'$*. Takes as input a long-term part of a master secret key* ltsk*, a constrained key* csk^**, and a constraint* C*. Outputs an updated ephemeral part of the master secret key* esk'*.*

Moreover, we require that correctness also holds with respect to $\mathsf{msk}' = (\mathsf{ltsk}, \mathsf{esk}')$ *and* csk^**, if* csk^* *is properly formatted. More formally, we have:*

Updatable correctness. *For all security parameters* λ*, all constraints* $C \in \mathcal{C}$*, all inputs* $x \in \mathcal{X}$*, and for all properly formatted* csk^* *and all* $\alpha \in \mathcal{S}$ *with* $C(x, \alpha) = 0$ *(authorized), it holds:*

$$\Pr\left[\begin{array}{l}(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathsf{esk}' \leftarrow \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*, C), \ : \ \mathsf{Eval}(\mathsf{msk}', x, \alpha) = \mathsf{CEval}(\mathsf{csk}^*, x) \\ \mathsf{msk}' \leftarrow (\mathsf{ltsk}, \mathsf{esk}')\end{array}\right] = 1 - \mathsf{negl}(\lambda),$$

Multi-instance Updatable Security. *For every polynomial* $\ell = \ell(\lambda) \in \mathsf{poly}(\lambda)$*, a* ShCPRF *is (1-key, selectively,* ℓ*-instance)-updatably secure if for all efficient adversaries* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following security experiment* $\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},b}(\lambda)$ *is negligible in* λ*. Here,* b *denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ*, the challenger*
   - *runs* $\mathcal{A}(1^\lambda)$ *and receives* ℓ *constraints* $C_i \in \mathcal{C}$ *and the (possibly corrupted) constrained keys* csk^*_i*, for all* $i \in [\ell]$*,*
   - *computes* $(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\mathsf{esk}'_i \leftarrow \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*_i, C_i)$*,*
   - *sets* $\mathsf{msk}_i := (\mathsf{ltsk}, \mathsf{esk}'_i)$*, and*
   - *samples a uniformly random function* $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$*, where* $\widetilde{\mathcal{F}}_\lambda$ *denotes the set of all functions from* $[\ell(\lambda)] \times \mathcal{X}_\lambda \times \mathcal{S}$ *to* $\mathcal{Y}$*.*
2. ***Evaluation queries:*** $\mathcal{A}$ *adaptively sends arbitrary inputs* $x \in \mathcal{X}$*, shifts* $\alpha \in \mathcal{S}$*, and index* $i \in [\ell]$ *to the challenger. For each triple* (x, α, i)*, if* $C_i(x, \alpha) = 0$*, then the challenger returns* $\perp$*. Otherwise, the challenger proceeds as follows:*
   - *If* $b = 0$*, it computes* $y \leftarrow \mathsf{Eval}(\mathsf{msk}_i, x, \alpha)$ *and returns* y*.*
   - *If* $b = 1$*, it computes* $y \leftarrow R(i, x, \alpha)$ *and returns* y*.*
3. ***Guess:*** $\mathcal{A}$ *outputs its guess* b'*, which is the output of the experiment.*

$\mathcal{A}$ *wins if* $b' = b$*, and its advantage* $\mathsf{Adv}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A}}(\lambda)$ *is defined as*

$$\mathsf{Adv}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ *and* KeyGen*.*

Pseudorandomness. *Define the following stateful oracle* $\mathcal{O}_{\mathsf{mi\text{-}eval}}$*:*

Oracle $\mathcal{O}_{\mathsf{mi\text{-}eval}}$

Initialize. Sample $\mathsf{msk}_i \leftarrow \mathsf{KeyGen}(1^\lambda)$, for all $i \in [\ell]$.

Evaluation. On input (i, x, α), return $\mathsf{Eval}(\mathsf{msk}_i, x, \alpha)$.

A ShCPRF is said to be multi-instance pseudorandom if for all efficient adversaries $\mathcal{A}$*, it holds that*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{mi\text{-}eval}}(\cdot,\cdot,\cdot)}(1^\lambda) = 1] - \Pr_{R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda}[\mathcal{A}^{R(\cdot,\cdot,\cdot)}(1^\lambda) = 1]\right| = 1 - \mathsf{negl}(\lambda),$$

where the left probability is over the randomness of KeyGen*, and* $\widetilde{\mathcal{F}}_\lambda$ *is the set of all functions from* $[\ell] \times \mathcal{X}_\lambda \times \mathcal{S}$ *to* $\mathcal{Y}$*.*

7.2 Constructing ℓ-instance updatably-secure shiftable CPRFs

We now describe a simple modification of the ShCPRF from Section 4 and prove that it satisfies updatable correctness and ℓ-instance updatable security. The construction is given in Figure 9.

Theorem 3. *If* $\mathcal{F}$ *is a family of RKA-secure pseudorandom functions with respect to affine related key derivation functions* Φ_{aff}*, as defined in Definition 12, then for every polynomial* $\ell(\lambda) \in \mathsf{poly}(\lambda)$*, Figure 9 instantiated with* $\mathcal{F}$ *is a (1-key, selectively,* ℓ*-instance)-updatably secure ShCPRF for inner-product predicates.*

Proof. Deferred to Appendix D.3. ■

Updatablity for the Ring-based ShCPRF for Inner Products

Public Parameters. Security parameter λ, finite ring $\mathcal{R}$ of order ℓ, integers m such that $m \geq \lambda$, vector length $n \geq 1$, and an RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ for affine RKA key-derivation functions.

$\mathsf{ShCPRF.KeyGen}(1^\lambda)$:

1 : $\mathsf{ltsk} := \Delta \xleftarrow{R} \mathcal{R}^m \setminus \{0\}$

2 : $\mathsf{esk} := (\mathbf{Z_0}, \mathbf{k_0}) \xleftarrow{R} \mathcal{R}^{m \times n} \times \mathcal{R}^m$

3 : **return** $\mathsf{msk} := (\mathsf{ltsk}, \mathsf{esk})$

$\mathsf{ShCPRF.Update}(\mathsf{ltsk}, \mathsf{csk}, \mathbf{z})$:

1 : **parse** $\mathsf{ltsk} =: \Delta$

2 : **parse** $\mathsf{csk} := (\mathbf{k'_0}, \mathbf{Z_1})$

3 : $\mathbf{Z'_0} := \mathbf{Z_1} + \Delta \mathbf{z}^\top$

4 : **return** $\mathsf{esk}' = (\mathbf{Z'_0}, \mathbf{k'_0})$

The remaining algorithms ($\mathsf{ShCPRF.Constrain}, \mathsf{ShCPRF.Eval}, \mathsf{ShCPRF.CEval}$), are as defined in Figure 1. Note that due to the now slightly different format of msk, the algorithms $\mathsf{ShCPRF.Constrain}$ and $\mathsf{ShCPRF.Eval}$ defined in Figure 1 need to be adapted to parse msk accordingly.

Fig. 9: A modified version of the KeyGen, and the additional Update algorithm for the ring-based shiftable CPRF framework from Figure 1.

7.3 Defining public-key PCFs for ListOT

In this section, we introduce the notion of public-key PCFs for ListOT. Our definition is geared towards our main application: non-interactive setup of pairwise OT channels over a large-scale network. Crucially for this application, our security notions incorporate the definition of *ℓ-instance security*, which say (informally) that if a user uses the *same* public key pk to establish an OT channel with ℓ different parties, then security is maintained even against an adversary corrupting all ℓ parties.

Definition 8 (ℓ-Instance Public-Key Pseudorandom Correlation Function for ListOT). *Let λ be a security parameter and $\lambda \leq n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Public-Key Pseudorandom Correlation Function (pkPCF) for ListOT is defined by a triple of algorithms* ($\mathsf{pkPCF.Gen}$, $\mathsf{pkPCF.KeyDer}$, $\mathsf{pkPCF.Eval}$) *with the following functionality. We leave the public parameters* PP *as an implicit input to all algorithms.*

- $\mathsf{pkPCF.Gen}(1^\lambda, \sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *Takes as input the security parameter λ and a party index σ. Outputs a public and private key pair $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$.*
- $\mathsf{pkPCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{\bar{\sigma}}) \to K_\sigma$. *Takes as input a party identifier $\sigma \in \{S, R\}$ (where $\bar{\sigma}$ denotes the other party's identifier), a secret key sk_σ, and a public key $\mathsf{pk}_{\bar{\sigma}}$. Outputs a key K_σ. We assume this algorithm is deterministic.*
- $\mathsf{pkPCF.Eval}(\sigma, K_\sigma, x) \to y_\sigma$. *Takes as input $\sigma \in \{S, R\}$, a key K_σ, and input $x \in \mathcal{X}$. Outputs a string $y_\sigma \in \{0,1\}^*$, where*
  - *if $\sigma = S$ then $y_\sigma = (L_0, L_1)$ for two lists L_0, L_1, and*
  - *if $\sigma = R$ then $y_\sigma = (b, v, \alpha)$ for a bit $b \in \{0,1\}$, a list entry $v \in L_0 \cup L_1$, and a lookup key α.*

We will use $\mathsf{pkPCF.EvalS}(K_S, x)$ and $\mathsf{pkPCF.EvalR}(K_R, x)$ as shorthand for the $\mathsf{pkPCF.Eval}$ algorithm used by the sender and receiver, respectively.

A public key PCF $\mathsf{pkPCF} = (\mathsf{pkPCF.Gen}, \mathsf{pkPCF.KeyDer}, \mathsf{pkPCF.Eval})$ *is a* (weak) ℓ-instance public-key PCF for ListOT*, if the following* correctness*, ℓ-instance sender security, and ℓ-instance receiver* security *properties hold. In each case, the adversary is given access to $N(\lambda) \in \mathsf{poly}(\lambda)$ samples.*

- ***Pseudorandomness.*** *For all efficient adversaries $\mathcal{A}$, and all $N \in \mathsf{poly}(\lambda)$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 10.

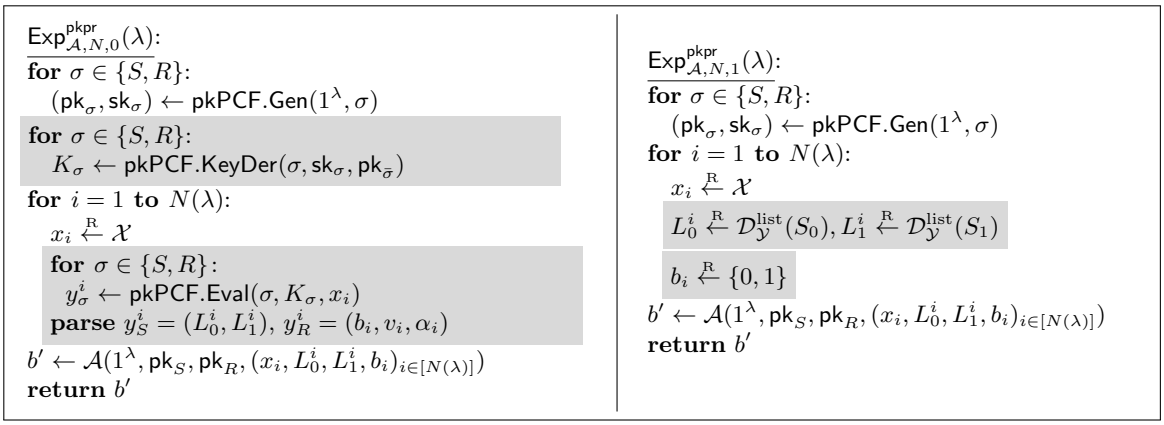

Fig. 10: (Partially) Pseudorandom outputs of a public-key PCF for ListOT.

– ***Correctness.*** *We want that for any $\lambda \in \mathbb{N}$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{ll} (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, \sigma) \text{ for } \sigma \in \{S,R\}, & \\ K_\sigma \leftarrow \mathsf{pkPCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{\bar{\sigma}}) \text{ for } \sigma \in \{S,R\}, & \\ x \xleftarrow{R} \mathcal{X}_\lambda, & : \ v \neq L_b[\alpha] \\ (L_0, L_1) \leftarrow \mathsf{pkPCF.EvalS}(K_S, x), & \\ (b, v, \alpha) \leftarrow \mathsf{pkPCF.EvalR}(K_R, x) & \end{array}\right],$$

i.e., that the (relevant) entry v is at position α of list L_b with a probability that is overwhelming in λ.

– ***ℓ-Instance Sender Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,0}(\lambda, \ell(\lambda)) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,1}(\lambda, \ell(\lambda)) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,b}(\lambda, \ell(\lambda))$, for $b \in \{0,1\}$, is as defined in Figure 11.

– ***ℓ-Instance Receiver Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,R,0}(\lambda, \ell(\lambda)) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,R,1}(\lambda, \ell(\lambda)) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,S,b}(\lambda, \ell(\lambda))$, for $b \in \{0,1\}$, is as defined in Figure 12.

7.4 Constructing public-key PCFs for ListOT

In this section, we provide a construction of public-key PCF for ListOT. In a nutshell, our construction is exactly the construction of PCF for ListOT (Figure 5) instantiated with an ℓ-instance updatable shiftable CPRF, enhanced with a distributed public-key setup protocol ($\mathsf{Gen}, \mathsf{KeyDer}$) to generate the ShCPRF keys. Formally, we define a distributed public-key setup protocol as follows:

Definition 9 (Distributed public-key setup protocol). *A distributed, corruptible public-key setup protocol* PKS *is parameterized by an updatable, shiftable CPRF* $\mathsf{ShCPRF} = (\mathsf{ShCPRF.KeyGen}, \mathsf{ShCPRF.Eval}, \mathsf{ShCPRF.Constrain}, \mathsf{ShCPRF.CEval}, \mathsf{ShCPRF.Update})$ *with constraint class $\mathcal{C}$, and consists of the following algorithms. We leave the public parameters* PP *as an implicit input to all algorithms.*

– $\mathsf{PKS.Gen}(1^\lambda, \sigma, m_\sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *Takes as input a party identifier $\sigma \in \{S,R\}$, a message m_σ where m_σ is a long-term secret key* ltsk *if $\sigma = S$ and a constraint $C \in \mathcal{C}$ if $\sigma = R$. Outputs a public-secret key pair.*

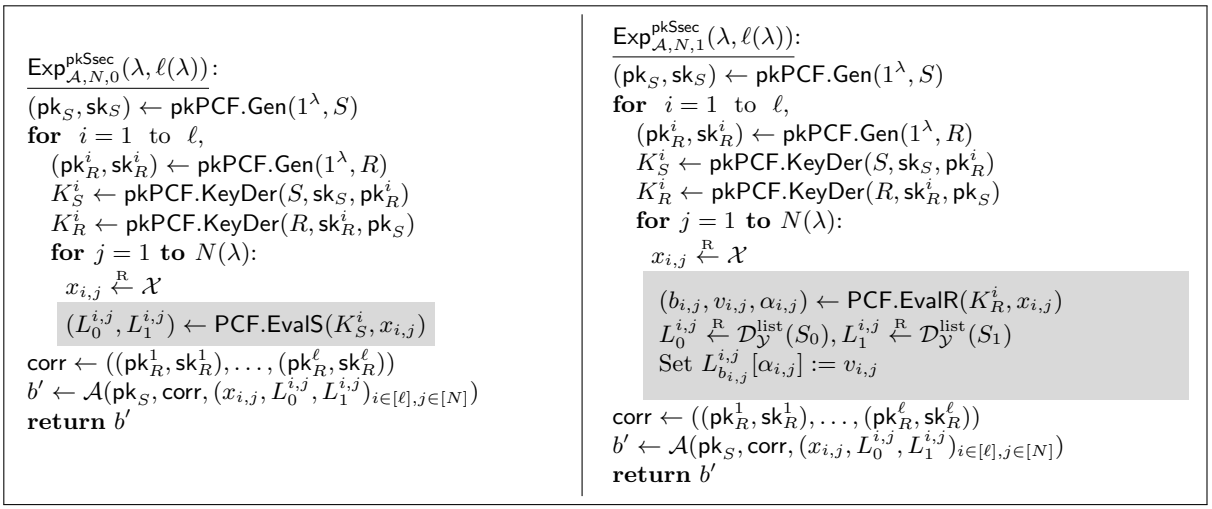

Fig. 11: ℓ-instance sender security game of a pkPCF for ListOT.

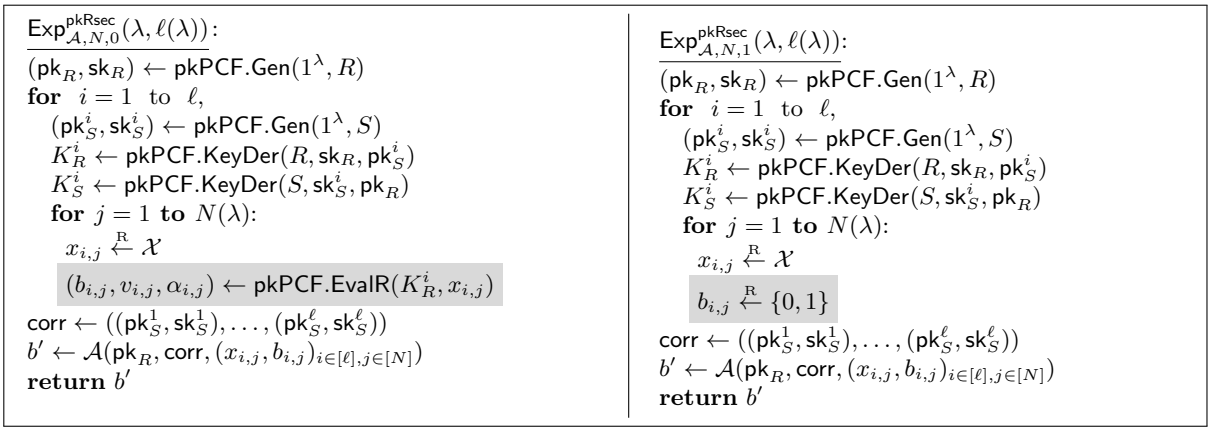

Fig. 12: ℓ-instance receiver security game of a pkPCF for ListOT.

– $\mathsf{PKS.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\overline{\sigma}})\to K_\sigma$. *Takes as input a party identifier* $\sigma\in\{S,R\}$*, the corresponding secret key* sk_σ*, and the other party's public key* $\mathsf{pk}_{\overline{\sigma}}$*. Outputs an evaluation key* K_σ*.*

We say $(\mathsf{PKS.Gen},\mathsf{PKS.KeyDer})$ *is a distributed corruptible public-key setup if the following one-message protocol realizes the ideal functionality described in Figure 13:*

1. *Each sender and receiver runs* $\mathsf{PKS.Gen}$ *and broadcasts the resulting public key.*
2. *Each sender (resp. receiver) uses the public key of each receiver (resp. sender) and its own secret key to derive a shared updatable ShCPRF key using* $\mathsf{PKS.KeyDer}$*.*

Theorem 4. *Let* $n=n(\lambda)\in\mathsf{poly}(\lambda)$ *and* $\mathcal{R}$ *be a finite ring of order* q*, with extension parameter* $m\geq\lambda$*. Let* $\mathsf{ShCPRF}=(\mathsf{KeyGen},\mathsf{Eval},\mathsf{Constrain},\mathsf{CEval},\mathsf{Update})$ *be an* ℓ*-instance updatable ShCPRF for inner-product predicates with domain* $\mathcal{R}^n$*, and* $f\colon\mathcal{R}^n\times\mathcal{R}^n\to\{0,1\}$ *a weak PRF family for inner-product membership with partitioning* $S_0\cup S_1=\mathcal{R}$*. Let* $\mathsf{PKS}=(\mathsf{Gen},\mathsf{KeyDer})$ *be a distributed public-key setup protocol for* ShCPRF*. Let* $\mathsf{pkPCF}=(\mathsf{Gen},\mathsf{KeyDer},\mathsf{Eval})$ *denote the PCF from Figure 5 (parametrized by* ShCPRF *as the ShCPRF), where the algorithm* $\mathsf{PCF.KeyGen}$ *is replaced by the distributed public-key setup algorithms:*

– $\mathsf{pkPCF.Gen}(1^\lambda,\sigma)$*. Takes as input a security parameter and party identifier* $\sigma\in\{S,R\}$*, and*

- *if* $\sigma=S$*, samples* $(\mathsf{ltsk},\mathsf{esk})\leftarrow\mathsf{ShCPRF.KeyGen}(1^\lambda)$ *and sets* $m_\sigma\leftarrow\mathsf{ltsk}$*,*
- *if* $\sigma=R$*, samples a random constraint* $\mathbf{z}$ *over* $\mathcal{R}^n$ *and sets* $m_\sigma\leftarrow\mathbf{z}$*.*

Outputs $(\mathsf{pk}_\sigma,\mathsf{sk}_\sigma)\leftarrow\mathsf{PKS.Gen}(1^\lambda,\sigma,m_\sigma)$*.*

– $\mathsf{pkPCF.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\bar{\sigma}}):=\mathsf{PKS.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\bar{\sigma}})$

Functionality $\mathcal{F}_{\mathsf{PKS}}$

Parameters. The ideal corruptible functionality $\mathcal{F}_{\mathsf{PKS}}$ is parameterized by an updatable shiftable CPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Update})$ that supports constraints in a constraint class $\mathcal{C}$.

Parties. An adversary $\mathcal{A}$, senders $S_1, \ldots, S_\ell$, and receivers $R_1, \ldots, R_\ell$

Procedure.
The functionality aborts if it receives any incorrectly formatted message.

Generation phase (one time).

1: Receive a message containing a long-term secret key part and an index (ltsk_i, i) from every sender S_i for all $i \in [\ell]$.
2: Receive a message containing a constraint and an index (C_j, j) from every receiver R_j for all $j \in [\ell]$.
3: Send ready to $\mathcal{A}$.

Key derivation phase (repeatable).

1: Receive a message (keyder, j) from a sender S_i and a message (keyder, i) from a receiver R_j for some $i, j \in [\ell]$.
2: Receive a message from $\mathcal{A}$ which is either empty, contains an ephemeral master secret key part esk, or contains a constrained key csk.
3: If $\mathcal{A}$ sent an empty message, i.e., the sender S_i and receiver R_j are both honest: sample esk uniformly as in KeyGen and compute $\mathsf{csk} \leftarrow \mathsf{Constrain}((\mathsf{ltsk}_i, \mathsf{esk}), C_j)$.
4: If $\mathcal{A}$ sent esk, i.e., the sender S_i is corrupted: compute $\mathsf{csk} \leftarrow \mathsf{Constrain}((\mathsf{ltsk}_i, \mathsf{esk}), C_j)$.
5: If $\mathcal{A}$ sent csk, i.e., receiver R_j is corrupted: compute $\mathsf{esk} \leftarrow \mathsf{Update}(\mathsf{ltsk}_i, \mathsf{csk}, C_j)$.
6: Output (esk, j) to S_i and (csk, i) to R_j.

Fig. 13: Corruptible ideal functionality for the distributed public key setup.

Then, pkPCF *is an ℓ-instance secure public-key PCF for ListOT.*

Proof. We consider each property in turn.

Pseudorandomness. We prove pseudorandomness via a sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid consists $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,0}(\lambda)$ from Figure 10, where $\mathsf{pkPCF.KeyGen}$, $\mathsf{pkPCF.KeyDer}$ and $\mathsf{pkPCF.Eval}$ are as defined in Theorem 4.

Hybrid $\mathcal{H}_1$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ sends an empty message to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of the ideal functionality, and receives no output. Then, it emulates the distribution of $(\mathsf{pk}_S, \mathsf{pk}_R)$. By security of the PKS scheme, $\mathcal{H}_0 \approx_c \mathcal{H}_1$. Note that from this hybrid, the distribution of $(\mathsf{pk}_S, \mathsf{pk}_R)$ is independent of $\mathbf{z}$ and msk.

Hybrid $\mathcal{H}_2$ (wPRF security). In this hybrid game, we replace each pseudorandom bit b_i sampled in $\mathcal{H}_1$ using the the IPM-wPRF f, with truly random bits sampled uniformly at random. The proof that $\mathcal{H}_2 \approx_c \mathcal{H}_1$ reduces to the pseudorandomness of f and is identical to the proof between hybrids $\mathcal{H}_1$ and $\mathcal{H}_0$ from the pseudorandomness proof of Theorem 2.

Hybrid $\mathcal{H}_3$ (ShCPRF pseudorandomness). In this hybrid, for all $i \in [N(\lambda)]$, the lists L^i_0, L^i_1 are sampled uniformly from $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(S_0)$ and $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(S_1)$, respectively, where the distribution $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(\cdot)$ is defined in Definition 5 and consists of uniformly random samples from $\mathcal{Y}$. The proof that $\mathcal{H}_3 \approx_c \mathcal{H}_2$ reduces to the pseudorandomness of ShCPRF and is identical to the proof between hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ in the pseudorandomness proof of Theorem 2. We observe that $\mathcal{H}_3$ is identical to $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,1}(\lambda)$, which concludes the proof of pseudorandomness.

Correctness. This is similar to correctness proof of Theorem 2. Observe that the entry v output by $\mathsf{PCF.EvalR}(K_R, x)$ is computed as $v \leftarrow \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$, where $\mathbf{x} \leftarrow \mathsf{map}(x)$ and csk is computed as in the key derivation phase of $\mathcal{F}_{\mathsf{PKS}}$ via $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$ for a constraint $\mathbf{z} \in \mathcal{R}^n$, if the adversary $\mathcal{A}$ is honest, or is a properly formatted csk as output by $\mathcal{A}$, in which case $\mathsf{esk}' \leftarrow \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}, \mathbf{z})$. In the first case ($\mathcal{A}$ is honest), correctness follows exactly as in the proof of Theorem 2, using the correctness of ShCPRF. Similarly, for the second case (with a properly formatted

csk given by $\mathcal{A}$), we make use of the updatable correctness of ShCPRF. Hence, using $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$, the entry v is equal to $\mathsf{Eval}(\mathsf{msk}', x, \alpha)$, where $\mathsf{msk}' \leftarrow (\mathsf{ltsk}, \mathsf{esk}')$ with $(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, for the (unique) $b' \in \{0,1\}$ with $\alpha \in S_{b'}$, we have that, with overwhelming probability, $L_{b'}[\alpha] = v$. As before, $b' = b$ by the property of the IPM-wPRF which guarantees that $b := f_{\mathbf{z}}(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_0$ and $f_{\mathbf{z}}(\mathbf{x}) = 1$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_1$.

Sender Security. We have a sequence of hybrids.

Hybrid $\mathcal{H}_0$. This consists of $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,0}(\lambda)$ defined in Figure 11, where $\mathsf{pkPCF.KeyGen}$, $\mathsf{pkPCF.KeyDer}$ and $\mathsf{pkPCF.Eval}$ are as defined in Theorem 4. In particular, we note that $\mathsf{pkPCF.EvalS}$ internally runs the updatable ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Update})$.

Hybrid $\mathcal{H}_1$. In this hybrid, for each $i \in [\ell(\lambda)]$ and $j \in [N(\lambda)]$, the call to $\mathsf{ShCPRF.Eval}(\mathsf{msk}_i, \mathbf{x}, \alpha)$ inside of the algorithm $\mathsf{pkPCF.EvalS}$ is replaced with a call to $\mathsf{ShCPRF.CEval}(\mathsf{csk}_i, \mathbf{x})$ whenever $(\mathbf{x}, \alpha)$ is authorized (i.e., $\langle \mathbf{z}_i, \mathbf{x} \rangle - \alpha = 0$), where csk_i is part of K_{R_i} in $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,0}(\lambda)$.

Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

Proof. By the updatable correctness of updatable ShCPRFs, we have that for all authorized $(\mathbf{x}_{i,j}, \alpha)$ pairs, $\mathsf{ShCPRF.Eval}(\mathsf{msk}_i, \mathbf{x}_{i,j}, \alpha) = \mathsf{ShCPRF.CEval}(\mathsf{csk}_i, \mathbf{x}_{i,j})$, with overwhelming probability. □

Hybrid $\mathcal{H}_2$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ simulates pk_S and extracts the constrained keys $(\mathsf{csk}_j)_{j \le \ell}$ computed by the ℓ corrupted receivers from $(\mathsf{pk}_S, \mathsf{sk}^j_R)$. It sends csk_j to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of $\mathcal{A}$ for each corrupted receiver R_j. Note that from this game onward, the updated master secret keys msk_j are not known anymore, which will allow us to invoke the multi-instance updatable security of ShCPRF in the next hybrid.

Hybrid $\mathcal{H}_3$. In this hybrid, the lists $L^{i,j}_0$, $L^{i,j}_1$ are sampled uniformly at random from $\mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_0), \mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_1)$, respectively. Then, for each (i,j) where $(\mathbf{x}_{i,j}, \alpha_{i,j})$ is an authorized pair, find the $b_{i,j} \in \{0,1\}$, such that $\alpha_{i,j} \in S_{b_{i,j}}$, and overwrite $L^{i,j}_{b_{i,j}}[\alpha_{i,j}] := \mathsf{CEval}(\mathsf{csk}_i, \mathbf{x}_{i,j})$.

Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$.

Proof. This follows from the ℓ-instance updatable security of ShCPRF; the proof proceeds essentially as the proof between $\mathcal{H}_3$ and $\mathcal{H}_2$ in the sender security proof of Theorem 4. □

Observe that $\mathcal{H}_3$ is exactly the experiment $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,1}(\lambda)$ experiment defined in Figure 11, which concludes the proof of sender security.

Receiver Security. We have a sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,0}(\lambda)$ experiment defined in Figure 12, where pkPCF is as defined in Theorem 4.

Hybrid $\mathcal{H}_1$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ simulates pk_R and extracts the ephemeral keys $(\mathsf{esk}_j)_{j \le \ell}$ computed by the ℓ corrupted senders from $(\mathsf{pk}_R, \mathsf{sk}^j_S)$. It sends esk_j to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of $\mathcal{A}$ for each corrupted sender S_j. Note that from this game onward, the constraint $\mathbf{z}$ is not known anymore, which will allow us to invoke the wPRF security in the next hybrid.

Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

Proof. This follows directly from the receiver security property of the PKS scheme, since pk_R is computationally indistinguishable from uniform. □

Hybrid $\mathcal{H}_2$. In this hybrid, we sample a uniformly random function R from the set of all functions from $\mathcal{X}_\lambda$ to $\{0,1\}$, and generate $b_{i,j} \leftarrow R(x_{i,j})$.

Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$.

Proof. The only difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is that $b_{i,j} = f_{\mathbf{z}}(x_{i,j})$ in $\mathcal{H}_1$, and $b_{i,j} = R(x_{i,j})$ in $\mathcal{H}_2$. As the $x_{i,j}$'s are uniformly random (and pk_R is simulated without using $\mathbf{z}$), any distinguisher between $\mathcal{H}_2$ and $\mathcal{H}_1$ immediately yields a distinguisher for the wPRF f, contradicting the pseudorandomness of $f_{\mathbf{z}}$. □

Hybrid $\mathcal{H}_3$. In this hybrid, each bit $b_{i,j}$ is sampled uniformly at random: $b_{i,j} \xleftarrow{R} \{0,1\}$. Note that this hybrid is exactly $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,1}(\lambda)$.

Claim. $\mathcal{H}_3 \approx_s \mathcal{H}_2$.

Proof. Since R is a truly random function, $\mathcal{H}_3$ and $\mathcal{H}_2$ are perfectly indistinguishable conditioned on all $x_{i,j}$'s being distinct. By a straightforward union bound, since all $x_{i,j}$'s are sampled randomly from $\mathcal{X}$, the condition is satisfied except with probability at most $(N \cdot \ell)^2/|\mathcal{X}_\lambda|$, which is negligible in λ, because $|\mathcal{X}_\lambda|$ is exponential in λ. □

This concludes the proof of receiver security, and the proof of Theorem 4. ■

7.5 Public-key setup from RingLWE

Protocol. Here we informally describe the distributed public-key setup protocol $(\mathsf{Gen}, \mathsf{KeyDer})$. See Figure 14 for a formal construction.

Public-Key Setup for Ring-based PCF for ListOT

Public Parameters. Security parameter λ, finite ring $\mathcal{R} = \mathbb{Z}_t$, vector length $n \geq 1$, integer m such that $m \geq \lambda$, polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$, noise distribution χ over $\mathcal{P}$, and random $a_0, a_1 \in \mathcal{P}$.

$\mathsf{Gen}(1^\lambda, S, \mathsf{ltsk} =: \Delta)$:

$s_0^1, \ldots, s_0^m \xleftarrow{R} \chi$
$\mathsf{sk}_S := (\Delta, s_0^1, \ldots, s_0^m)$
$e_0^1, \ldots, e_0^m \xleftarrow{R} \chi$
foreach $i \in [m]$:
  $\mathsf{pk}_S^i := \Delta_i \cdot a_0 + s_0^i a_1 + e_0^i$
$\mathbf{k}_0 \xleftarrow{R} \mathcal{R}^m$
return $(\mathsf{pk}_S := (\mathbf{k}_0, (\mathsf{pk}_S^i)_{i \leq m}), \mathsf{sk}_S)$

$\mathsf{Gen}(1^\lambda, R, C_{\mathbf{z}})$:

parse $\mathbf{z} \in \mathcal{R}^n$ from $C_{\mathbf{z}}$
$s_1 \xleftarrow{R} \chi$
$\mathsf{sk}_R := (\mathbf{z}, s_1)$
let $z \in \mathcal{P}$ have coeffs. $\frac{q}{t} \cdot (\mathbf{z} \| 0^{\eta - n})$
$e_1, e_1' \xleftarrow{R} \chi$
$\mathsf{pk}_R := (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$
return $(\mathsf{pk}_R, \mathsf{sk}_R)$

$\mathsf{KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_R)$:

parse $\mathsf{sk}_S = (\Delta, s_0^1, \ldots, s_0^m)$
foreach $i \in [m]$:
  $z_0^i \leftarrow \left\langle \mathsf{pk}_R, \left(\Delta_i, s_0^i\right) \right\rangle$
  parse $z_0^i \in \mathcal{P}$ as $\tilde{\mathbf{z}}_0^i \in \mathbb{Z}_q^n$
  round $\mathbf{z}_0^i = \left\lceil \tilde{\mathbf{z}}_0^i \right\rfloor_t \in \mathbb{Z}_t^n$
$\mathsf{esk} = (\mathbf{k}_0, \mathbf{Z}_0 = (\mathbf{z}_0^i)_{i \leq m})$
return $K_S := (\Delta, \mathsf{esk})$

$\mathsf{KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_S)$:

parse $\mathsf{sk}_R = (\mathbf{z}, s_1)$, $\mathsf{pk}_S := (\mathbf{k}_0, (\mathsf{pk}_S^i)_{i \leq m})$
foreach $i \in [m]$:
  $z_1^i \leftarrow \mathsf{pk}_S^i \cdot s_1$
  parse z_1^i as $\tilde{\mathbf{z}}_1^i \in \mathbb{Z}_q^n$
  round $\mathbf{z}_1^i = \left\lceil \tilde{\mathbf{z}}_1^i \right\rfloor_t \in \mathbb{Z}_t^n$
$\mathsf{csk} := (\mathbf{k}_0, \mathbf{Z}_1 := (\mathbf{z}_1^i)_{i \leq m})$
return $K_R := (\mathsf{csk}, \mathbf{z})$

Fig. 14: A distributed public-key setup protocol for the ring-based updatable ShCPRF for inner products. Note that when parsing elements of $\mathcal{P}$ as vectors in $\mathbb{Z}_q^n$, we ignore the last $\eta - n$ coefficients.

We assume $\mathcal{R} = \mathbb{Z}_t$ and work over a polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ (we will define the RingLWE parameters η, q later). The public parameters include random $a_0, a_1 \in \mathcal{P}$. In the public key generation phase, the sender samples Δ from $\mathcal{R}^m \setminus \{0\}$, m secrets $s_0^1, \ldots, s_0^m$ from a noise distribution, and noise $e_0^1, \ldots, e_0^m$. It publishes as its public key $\mathsf{pk}_S = (\Delta_i \cdot a_0 + s_0^i a_1 + e_0^i)_{i \in [m]}$, along with $\mathbf{k}_0$, which it samples randomly from $\mathcal{R}^m$.

The receiver samples $\mathbf{z}$, a secret s_1 from the noise distribution, and noise e_1, e_1'. It encodes $\frac{q}{t} \cdot \mathbf{z}$ in a polynomial $z \in \mathcal{P}$ and publishes as its public key $\mathsf{pk}_R = (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$.

In the evaluation key derivation phase, the sender computes, for each $i \in [m]$, the rounded inner product $\lceil \langle \mathsf{pk}_R, (\Delta_i, s_0^i) \rangle \rfloor_t$[13] and parses the resulting m polynomials as a matrix in $\mathbb{Z}_t^{m \times n}$ (ignoring the last $\eta - n$ coefficients when parsing polynomial ring elements as vectors). The receiver computes, for each $i \in [m]$, the rounded product $\lceil \mathsf{pk}_S^i \cdot s_1 \rfloor_t$ and, as before, parses the result as a matrix in $\mathbb{Z}_t^{m \times n}$.

Concrete parameters. We rely on the RingLWE assumption (with a superpolynomial modulus-to-noise ratio) in the polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ for some η a power of 2. We use the normal form (see Definition 2), where the secret is drawn from the noise distribution rather than uniformly. This is a standard choice in practical RingLWE-based schemes with hardness supported by security reductions [58, 2, 60, 41].

We use a standard choice of error distribution, a discrete Gaussian with standard deviation $\sigma = 3.2$. When applying the rounding lemma (Lemma 1) to prove correctness of our protocol, note that the failure probability of rounding an element of $\mathbb{Z}_q$ to an element of $\mathcal{R} = \mathbb{Z}_t$ will grow with $\frac{t \cdot B}{q}$, where B is a bound on the magnitude of the error term in the derived polynomials (that depends on χ and η). To ensure correctness with overwhelming probability, *i.e.*, that with probability at least $1 - 2^{40}$, the sender and receiver correctly round all $n \cdot m$ coefficients, we will set $q = t \cdot B \cdot n \cdot m \cdot 2^{40}$.

We need that $\eta \geq n$ (for values of n chosen for both the BIPSW and GAR instantiations), and following post-quantum security standards for normal form RingLWE [2], the choice of $\eta = 2^{12}$ can support up to a 103-bit modulus q, which is more than sufficient for our choice of q.

Public key size. The sender's public key consists of m elements of $\mathcal{P}$, as well as $\mathbf{k}_0 \in \mathcal{R}^m$, so we can bound the key size by $m \cdot (\eta \cdot \log q + \log t)$ bits. The sender's public key is roughly 5.5 MB in size for the BIPSW IPM-wPRF and GAR IPM-wPRF. However, the receiver's public key only consists of 2 elements of $\mathcal{P}$, which makes it roughly 85 kB in size.

Theorem 5. *The distributed public-key setup protocol* $(\mathsf{PKS.Gen}, \mathsf{PKS.KeyDer})$ *defined in Figure 14 securely implements the corruptible ideal functionality represented in Figure 13 with respect to the ring-based updatable* ShCPRF *framework defined in Figures 1 and 9.*

Proof. Deferred to Appendix D.4. ■

8 Implementation and Evaluation

Implementation. We implement QuietOT in C with roughly 1200 lines of code for the BIPSW and GAR implementations combined. Our implementation is open source and available at

https://github.com/sachaservan/QuietOT.

We additionally implement the BCMPR silent OT protocol in C in roughly 600 lines of code using the P-256 elliptic curve implementation available in the OpenSSL library [72]. For OSY, we estimate their runtime by benchmarking the dominant cost of their construction: computing $\lambda = 128$ modular exponentiations in a 3200-bit RSA group. To heuristically instantiate the random oracle $H(\cdot)$, we use fixed-key AES, operating with 128-bit inputs, and truncate the output to a single bit (such an instantiation is shown to be correlation-robust in the idea cipher model [49]). To generate pseudorandom inputs for the PCF, we stretch a short seed (common to both the sender and receiver) using AES in CTR mode. Our implementations make use of several optimizations, which are described further in Section 8.1. We use the state-of-the-art implementation of existing OT extension protocols (IKNP, SoftSpoken, RRT) available in libOTe [68] to compare to other OT extension protocols. In order to provide a fairer comparison to existing OT extension protocols, we do *not* include the base OT costs required in SoftSpoken and IKNP.

Environment. We run our benchmarks on an AWS `c5.metal` and `t2.small` instances, and on an Apple M1 Pro laptop computer, using a single core. Because network latency and bandwidth can fluctuate leading to high variance, our benchmarks take into account only the processing time

[13] Recall that we write $\lceil \cdot \rfloor_t$ to denote "rounding" a polynomial coefficient-by-coefficient.

required by the sender and receiver. We compare the network overhead between each protocol using the "bits/OT" measure, which provides an objective and consistent comparison between protocols, avoiding network-specific or implementation differences.[14]

Parameters. We fix the security parameter $\lambda = 128$. For BIPSW, we set $n = 768$ and pre-compute inner products with CPRF keys in blocks of 16 bits (see Section 8.1). We operate over the ring $\mathbb{Z}_6$, which allows us to use CRT decomposition and pack 128 elements of $\mathbb{Z}_2$ into one machine word. For GAR, we set $n = 2048$, $\ell_1 = 5$ and $\ell_2 = 15$. This allows us to work over the ring $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_{16}$. The choice of $n = 2048$ is very conservative but allows us to sample indices in $\{1, \ldots, 2048\}$ efficiently without rejection sampling. In turn, this improves concrete performance by allowing us to efficiently generate inputs for the wPRF (all we require is checking that the sampled set of random indices consist of distinct elements). SoftSpoken has a tunable tradeoff between communication and computational efficiency parameterized by k. For a given k SoftSpoken requires λ/k communication but increases computation by a factor of $2^k/k$. Small values of k (e.g., $k = 4$) provide a good tradeoff in practice, resulting in 32 bits/OT at an increase of $4\times$ in computation.

Communication costs and comparison. QuietOT with BIPSW as the IPM-wPRF requires 7 bits of communication per chosen-bit OT. For random choice bits, communication is only 6 bits since the receiver does not need to send its masked bit. Moreover, for random OT (when the sender inputs are also random), the messages m_0 and m_1 can be set to the first elements of L_0 and L_1, respectively, reducing communication to $|S_0| + |S_1| - 2$ (or 4 bits when using the BIPSW IPM-wPRF). QuietOT with GAR as the IPM-wPRF requires 33 bits of communication per chosen-bit OT. However, the same logic above reduces communication to 32 bits/OT when the choice bit is random and 30 bits/OT for random OT. QuietOT beats SoftSpoken on communication (for reasonable choices of k) when instantiated with BIPSW and remains on-par with SoftSpoken in terms of communication when instantiated with GAR. Silent OT protocols (i.e., RRT, BCMPR, OSY) have an optimal 3 bits/OT of communication and 2 bits/OT when the receiver's choice bit is random. This makes QuietOT roughly 2-10$\times$ worse in terms of communication when compared to Silent OT.

	$\lvert\mathsf{pk}_{\mathsf{sender}}\rvert$	$\lvert\mathsf{pk}_{\mathsf{receiver}}\rvert$	**OT/s** (M1 Pro)	**OT/s** (c5.metal)	**OT/s** (t2.small)	**Bits/OT**
IKNP			2,592,000	34,174,000	12,264,000	128
SoftSpoken ($k = 2$)			2,732,000	52,676,000	33,121,000	64
SoftSpoken ($k = 4$)			1,636,000	44,443,000	27,504,000	32
SoftSpoken ($k = 8$)			249,000	9,500,000	5,891,000	16
SoftSpoken ($k = 16$)			2,000	76,000	49,000	8
RRT			1,230,000	6,856,000	2,492,000	3
OSY	50 kB	1 kB	0.6	0.5	0.3	3
BCMPR (BIPSW)	63 kB	72 kB	15,000	12,000	8,000	3
BCMPR (GAR)	33 kB	38 kB	21,000	17,000	11,000	3
QuietOT (BIPSW)	5.4 MB	84 kB	1,165,000	561,000	362,000	7
with AVX512 support			N/A	1,265,000	N/A	7
QuietOT (GAR)	5.6 MB	88 kB	1,198,000	526,000	336,000	33

Table 2: OTs per second on a single core generated by the sender. Note that libOTe is not optimized for M1 since the AVX instructions are not available on M1 processors, hence we report these numbers in gray. The GAR IPM-wPRF cannot benefit from AVX due to limited bit-slicing opportunities. Setup costs are excluded.

Computational costs and comparison. The state-of-the-art OT extension protocol is SoftSpoken. To provide an apples-to-apples comparison of the computational costs while fixing the communication overhead in SoftSpoken, we could set $k = 18$ and $k = 4$ in SoftSpoken, leading to 7.1 bits/OT and

[14] The libOTe implementation is evaluated on `localhost`, and therefore is somewhat limited by the kernel when transferring data making IKNP slower than SoftSpoken.

32 bits/OT, respectively. However, SoftSpoken becomes very inefficient with large k which does not make the comparison fair when QuietOT is instantiated using BIPSW. Comparing to SoftSpoken with small k and QuietOT (when instantiated with either BIPSW or GAR) shows that QuietOT is roughly one to two orders of magnitude slower. However, we stress that SoftSpoken benefits a lot more from advanced hardware instructions than QuietOT, potentially making QuietOT outshine SoftSpoken on weak(er) devices. This is evidenced by QuietOT outperforming the SoftSpoken implementation on the M1 (where AVX512 is not available). Unfortunately, since the libOTe implementation is not optimized for performance when AVX is disabled, performing a head-to-head comparison difficult. Comparing QuietOT to BCMPR (state-of-the-art *public-key* OT protocol) shows that QuietOT is up to 100× faster in terms of computation while only increasing communication by a few bits.

Public key size. Our public key setup has public keys that are roughly 20 to 60 times larger compared to the public keys in BCMPR and OSY. This is primarily due to the parameters required for the RingLWE assumption (see Section 7.5). However, as a consequence, we obtain plausible post-quantum security. In practical terms, however, the average web page size is roughly 2 MB as of 2023 [52], making the overall key size very reasonable for use on the Internet. Additionally, we note that this is the *sender*'s public key size—the receiver's public key in our construction is only around 90 kB, which could potentially be beneficial to some applications.

8.1 Optimizations in the implementation

Our implementation makes use of several optimizations which we detail here.

Optimizing the BIPSW instantiation. We make several observations allowing us to concretely optimize the BIPSW instantiation from Figure 7.

Working over the CRT decomposition. All computations over $\mathcal{R} = \mathbb{Z}_6$ can we computed over the CRT decomposition $\mathbb{Z}_2 \times \mathbb{Z}_3 \simeq \mathbb{Z}_6$. This enables applying the following two optimizations:

- *Bit-sliced arithmetic in* $\mathbb{Z}_2$. We can pack the m elements of $\mathbb{Z}_2$ into $\lfloor m/64 \rfloor$ machine words (on 64-bit word processors). This allows parallelizing the $\mathbb{Z}_2$ component of all operations over $\mathcal{R}^m$ by using a single machine instruction for each batch of $\lfloor m/64 \rfloor$ elements of $\mathcal{R}^m$.
- *Bit-sliced arithmetic in* $\mathbb{Z}_3$. While we can also pack the $\mathbb{Z}_3$ elements into $\lfloor 2m/64 \rfloor$ machine words (using two bits for each element of $\mathbb{Z}_3$ on a 64-bit machine), such a packing requires computing operations in $\mathbb{Z}_3$ over the bit-sliced representation. Fortunately, this very problem was explored in WAVE [10, Appendix B.1], where they provide an efficient bit-sliced representation for computing fast bit-sliced arithmetic over $\mathbb{Z}_3$. In particular, each arithmetic operation in $\mathbb{Z}_3$ requires using only 7 machine instructions.

Our implementation in C. Concretely, we pack ring elements into `uint128_t` types in C (which represent two 64-bit machine words). This allows us to pack the 128 $\mathbb{Z}_2$ elements into one `uint128_t` type and pack the high and low order bits of the 128 $\mathbb{Z}_3$ elements into two `uint128_t` types. We can then perform bit-sliced arithmetic over this packed representation.

Preprocessing inner products. A separate optimization, which we find also applies to the construction of BCMPR (when instantiated with the BIPSW IPM-wPRF), is to preprocess the inner products over small chunks of the key. When using the BIPSW wPRF, each matrix-vector product $\mathbf{Z_0 x}$ can be equivalently written as a sum of smaller matrix-vector products. More precisely, let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n/t})$ such that $|\mathbf{x}_i| \in \{0,1\}^t$ (we assume that t divides n) and let $\mathbf{Z_0} = (\mathbf{Z_0}_1, \ldots, \mathbf{Z_0}_{n/t})$. Then, by preprocessing all possible 2^t matrix-vector products associated with the i-th column block matrix $\mathbf{Z_0}_i$ and storing the results, we can efficiently look up the result for any input block $\mathbf{x}_i$, saving a factor t in computation at a cost of 2^t in extra storage.

Optimizing the GAR instantiation. Unfortunately, we find fewer ways to optimize the GAR instantation compared to our BIPSW instantiation. In particular, the GAR instantiation does not benefit from preprocessing opportunities that we identify for our BIPSW instantiation. However, we note that we can still take advantage of bit-sliced operations to compute the m operations over $\mathbb{Z}_2$ in parallel. Performing bit-sliced arithmetic over $\mathbb{Z}_{16}$ (when $\ell = 16$), in contrast, is more challenging.

Fast arithmetic over $\mathbb{Z}_{16}$. We found it to be more efficient to *not* pack the $\mathbb{Z}_{16}$ elements and instead just use one byte for each of the 128 elements of $\mathbb{Z}_{16}^{128}$. This allows us to sum modulo 16 by first

computing the sum over the integers and then using a bit-mask to reduce modulo 16. In particular, we can sum two elements of $\mathbb{Z}_{16}$ via integer addition (summing two bytes) followed by a bit-wise AND with `0b00001111`, which zeroes-out the carry bit. This optimization makes summation modulo 16 fast, mitigating the impact of not being able to perform bit-sliced arithmetic for the $\mathbb{Z}_{16}$ elements.

General optimization: Compressing hash inputs via universal hashing. In the ring element representations of both the BIPSW and GAR instantiations, we end up with a tightly packed representation of the $\mathbb{Z}_2$ elements but only a "loosely-packed" representation of the $\mathbb{Z}_3$ elements (resp. $\mathbb{Z}_{16}$ elements). The naive approach would be to feed the entire bit-string representation of the ring elements (the ShCPRF key) and input $\mathbf{x}$ into the random oracle, which is heuristically instantiated using fixed-key AES [49]. However, this would require breaking up the input string into blocks of 128 bits and then xoring all the resulting AES outputs together. While this solution does not introduce noticeable slowdowns for the BIPSW instantiation,[15] it is not ideal for the GAR instantiation. For the GAR implementation, this approach would require packing all the elements of $\mathbb{Z}_{16}^{128}$ into four AES blocks which is inefficient since the packing itself is slow (recall that each element is represented as a byte for fast arithmetic operations). However, we additionally need to pack the ShCPRF input $\mathbf{x}$, which would lead to even more overheads. To avoid these inefficiencies, we instead choose to compress the representation of the ring elements and input $\mathbf{x}$ into tightly packed λ-bit strings by using a universal hash, which acts as a randomness extractor for the input to the random oracle. Specifically, we can make use of the leftover hash lemma [50] to extract $\lambda \approx 128$ bits from the representation of the ring elements. This allows us to then only perform one AES call, using the compressed 128-bit representation as input. We do the same for the ShCPRF input $\mathbf{x}$ and the $\mathbb{Z}_2^{128}$ block, leading to a total of three independent AES calls that we then truncate and xor together. The standard LHL bound requires $128 + 2\kappa$ bits of entropy to extract 128-bits that are statistically close to uniform in the worst-case [50], where κ is a statistical security parameter. However, the generalized LHL bound of Barak et al. [12] allows us to do better. Specifically, they prove that when extracting randomness to use as a key for a *weak* PRF (we assume that our ShCPRF takes uniformly random inputs and thus is a weak PRF)[16] the LHL bound can be improved to $128 + \kappa$, which saves a factor of two in the entropy loss. Therefore, we can increase the ring dimension m to $128 + 64$ to ensure the universal hashing produces a near-uniform 128 bit key for the ShCPRF, with ≥ 64-bits of *statistical* security. As for the input $\mathbf{x}$, universal hashing provides 128-bits with even more statistical security since under our concrete parameter choice for GAR, we already have 308 bits of entropy in the input $\mathbf{x}$, which already give more than 64-bits of statistical security under the basic LHL bound. Finally, to further improve efficiency, we use an *almost*-universal (as opposed to perfectly universal) hash function, which results in faster implementations while only sacrificing a few bits of statistical security [12]. In our implementation of the GAR instantiation, we use the open-source `Polymur` [65] almost-universal hash function for this purpose.

Acknowledgements

We thank the anonymous reviewers for helpful comments and pointing out a flaw in a previous version of our ShCPRF parameter choices. We thank Peter Rindal for help with running the libOTe library. We thank Sheela Devadas for helpful discussions on RingLWE. Geoffroy Couteau and Alexander Koch acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE) and under the France 2030 ANR Project ANR-22-PECY-003 SecureCompute. This work is supported by ERC grant OBELiSC (101115790), and NSF grant 2330065.

[15] In particular, we only need ≈ 80 bits for the $\mathbb{Z}_3$ components (which can be represented with two 128-bit blocks, leaving 96 bits "on-the-table" into which we can pack the ShCPRF input string. This leads to only three AES calls to instantiate the random oracle.

[16] We can view the inputs to the ShCPRF as having ≥ 128 bits of entropy—even if they are potentially non-uniform—since the input to the ShCPRF in our framework is itself *an input to a weak PRF*.

References

[1] M. Abdalla, F. Benhamouda, A. Passelègue, and K. G. Paterson. "Related-key security for pseudorandom functions beyond the linear barrier". In: *CRYPTO 2014*. Ed. by J. A. Garay and R. Gennaro. LNCS 8616. Springer, 2014, pp. 77–94. DOI: 10.1007/978-3-662-44371-2_5.

[2] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. *Homomorphic Encryption Security Standard*. Tech. rep. HomomorphicEncryption.org, 2018. URL: https://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf.

[3] B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. "Secure arithmetic computation with constant computational overhead". In: *CRYPTO 2017*. Ed. by J. Katz and H. Shacham. LNCS 10401. Springer, 2017, pp. 223–254. DOI: 10.1007/978-3-319-63688-7_8.

[4] B. Applebaum, D. Harnik, and Y. Ishai. "Semantic security under related-key attacks and applications". In: (2011). Ed. by B. Chazelle, pp. 45–60. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/30.html.

[5] B. Applebaum and S. Lovett. "Algebraic attacks against random local functions and their countermeasures". In: *STOC 2016*. Ed. by D. Wichs and Y. Mansour. ACM, 2016, pp. 1087–1100. DOI: 10.1145/2897518.2897554.

[6] B. Applebaum and P. Raykov. "Fast pseudorandom functions based on expander graphs". In: *TCC 2016-B*. Ed. by M. Hirt and A. D. Smith. LNCS 9985. Springer, 2016, pp. 27–56. DOI: 10.1007/978-3-662-53641-4_2.

[7] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. "More efficient oblivious transfer and extensions for faster secure computation". In: *CCS 2013*. Ed. by A. Sadeghi, V. D. Gligor, and M. Yung. ACM, 2013, pp. 535–548. DOI: 10.1145/2508859.2516738.

[8] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. "More efficient oblivious transfer extensions with security for malicious adversaries". In: *EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. LNCS 9056. Springer. 2015, pp. 673–701. DOI: 10.1007/978-3-662-46800-5_26.

[9] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. "Constrained PRFs for NC^1 in traditional groups". In: *CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. LNCS 10992. Springer, 2018, pp. 543–574. DOI: 10.1007/978-3-319-96881-0_19.

[10] G. Banegas, K. Carrier, A. Chailloux, A. Couvreur, T. Debris-Alazard, P. Gaborit, P. Karpman, J. Loyer, R. Niederhagen, N. Sendrier, B. Smith, and J.-P. Tilich. *WAVE: Round 1 Submission*. Version 1. 2023. URL: https://wave-sign.org/wave_documentation.pdf (visited on 01/21/2024).

[11] A. Banerjee, C. Peikert, and A. Rosen. "Pseudorandom functions and lattices". In: *EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. LNCS 7237. Springer, 2012, pp. 719–737. DOI: 10.1007/978-3-642-29011-4_42.

[12] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu. "Leftover Hash Lemma, Revisited". In: *CRYPTO 2011*. Ed. by P. Rogaway. LNCS 6841. Springer, 2011, pp. 1–20. DOI: 10.1007/978-3-642-22792-9_1.

[13] J. Bartusek, S. Garg, D. Masny, and P. Mukherjee. "Reusable two-round MPC from DDH". In: *TCC 2020*. Ed. by R. Pass and K. Pietrzak. LNCS 12551. Springer, 2020, pp. 320–348. DOI: 10.1007/978-3-030-64378-2_12.

[14] D. Beaver. "Correlated pseudorandomness and the complexity of private computations". In: *STOC 1996*. Ed. by G. L. Miller. ACM, 1996, pp. 479–488. DOI: 10.1145/237814.237996.

[15] D. Beaver. "Precomputing oblivious transfer". In: *CRYPTO 1995*. Ed. by D. Coppersmith. LNCS 963. Springer, 1995, pp. 97–109. DOI: 10.1007/3-540-44750-4_8.

[16] M. Bellare and T. Kohno. "A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications". In: *EUROCRYPT 2003*. Ed. by E. Biham. LNCS 2656. Springer, 2003, pp. 491–506. DOI: 10.1007/3-540-39200-9_31.

[17] M. Bellare and P. Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *CCS 1993*. 1993, pp. 62–73. DOI: 10.1201/9781420010756.

[18] E. Biham. "New types of cryptanalytic attacks using related keys". In: *EUROCRYPT 1993*. Ed. by T. Helleseth. LNCS 765. Springer, 1994, pp. 398–409. DOI: 10.1007/3-540-48285-7_34.

[19] D. Boneh, Y. Ishai, A. Passelègue, A. Sahai, and D. J. Wu. "Exploring crypto dark matter: New simple PRF candidates and their applications". In: *TCC 2018*. Ed. by A. Beimel and S. Dziembowski. LNCS 11240. Springer, 2018, pp. 699–729. DOI: 10.1007/978-3-030-03810-6_25.

[20] D. Boneh and B. Waters. "Constrained pseudorandom functions and their applications". In: *ASIACRYPT 2013*. Ed. by K. Sako and P. Sarkar. LNCS 8270. Springer, 2013, pp. 280–300. DOI: 10.1007/978-3-642-42045-0_15.

[21] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl. "Correlated pseudorandomness from expand-accumulate codes". In: *CRYPTO 2022*. Ed. by Y. Dodis and T. Shrimpton. LNCS 13508. Springer, 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21.

[22] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. "Efficient two-round OT extension and silent non-interactive secure computation". In: *CCS 2019*. Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM, 2019, pp. 291–308. DOI: 10.1145/3319535.3354255.

[23] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. "Correlated pseudorandom functions from variable-density LPN". In: *FOCS 2020*. Ed. by S. Irani. IEEE, 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103.

[24] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. "Efficient pseudorandom correlation generators: Silent OT extension and more". In: *CRYPTO 2019*. Ed. by A. Boldyreva and D. Micciancio. LNCS 11694. Springer, 2019, pp. 489–518. DOI: 10.1007/978-3-030-26954-8_16.

[25] E. Boyle, S. Goldwasser, and I. Ivan. "Functional signatures and pseudorandom functions". In: *PKC 2014*. Ed. by H. Krawczyk. LNCS 8383. Springer, 2014, pp. 501–519. DOI: 10.1007/978-3-642-54631-0_29.

[26] E. Boyle, L. Kohl, and P. Scholl. "Homomorphic secret sharing from lattices without FHE". In: *EUROCRYPT 2019*. Ed. by Y. Ishai and V. Rijmen. LNCS 11477. Springer, 2019, pp. 3–33. DOI: 10.1007/978-3-030-17656-3_1.

[27] Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. "Private constrained PRFs (and more) from LWE". In: *TCC 2017*. Ed. by Y. Kalai and L. Reyzin. LNCS 10677. Springer, 2017, pp. 264–302. DOI: 10.1007/978-3-319-70500-2_10.

[28] Z. Brakerski and V. Vaikuntanathan. "Constrained Key-Homomorphic PRFs from Standard Lattice Assumptions: Or: How to Secretly Embed a Circuit in Your PRF". In: *TCC 2015*. Ed. by Y. Dodis and J. B. Nielsen. LNCS 9015. Springer, 2015, pp. 1–30. DOI: 10.1007/978-3-662-46497-7_1.

[29] C. Brzuska, G. Couteau, C. Egger, P. Karanko, and P. Meyer. *New random oracle instantiations from extremely lossy functions*. 2023. Cryptology ePrint Archive, Report 2023/1145.

[30] D. Bui, G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. "Fast Public-Key Silent OT and More from Constrained Naor-Reingold". In: *EUROCRYPT 2024*. Ed. by M. Joye and G. Leander. LNCS 14656. Springer, 2024, pp. 88–118. DOI: 10.1007/978-3-031-58751-1_4.

[31] R. Canetti and Y. Chen. "Constraint-hiding constrained PRFs for NC^1 from LWE". In: *EUROCRYPT 2017*. Ed. by J. Coron and J. B. Nielsen. LNCS 10210. Springer, 2017, pp. 446–476. DOI: 10.1007/978-3-319-56620-7_16.

[32] R. Canetti and M. Fischlin. "Universally Composable Commitments". In: *CRYPTO 2001*. Ed. by J. Kilian. LNCS 2139. Springer, 2001, pp. 19–40. DOI: 10.1007/3-540-44647-8_2.

[33] Y. Chen, V. Vaikuntanathan, and H. Wee. "GGH15 beyond permutation branching programs: proofs, attacks, and candidates". In: *CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. LNCS 10992. Springer, 2018, pp. 577–607. DOI: 10.1007/978-3-319-96881-0_20.

[34] M. Ciampi, R. Ostrovsky, L. Siniscalchi, and H. Waldner. "List oblivious transfer and applications to round-optimal black-box multiparty coin tossing". In: *CRYPTO 2023*. Ed. by H. Handschuh and A. Lysyanskaya. LNCS 14081. Springer, 2023, pp. 459–488. DOI: 10.1007/978-3-031-38557-5_15.

[35] G. Couteau and C. Ducros. "Pseudorandom Correlation Functions from Variable-Density LPN, Revisited". In: *PKC 2023*. Ed. by A. Boldyreva and V. Kolesnikov. LNCS 13941. Springer, 2023, pp. 221–250. DOI: 10.1007/978-3-031-31371-4_8.

[36] G. Couteau, A. Dupin, P. Méaux, M. Rossi, and Y. Rotella. "On the concrete security of Goldreich's pseudorandom generator". In: *ASIACRYPT 2018*. Ed. by T. Peyrin and S. D. Galbraith. LNCS 11273. Springer, 2018, pp. 96–124. DOI: 10.1007/978-3-030-03329-3_4.

[37] G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. "Constrained Pseudorandom Functions from Homomorphic Secret Sharing". In: *EUROCRYPT 2023*. Ed. by C. Hazay and M. Stam. LNCS 14006. Springer, 2023, pp. 194–224. DOI: 10.1007/978-3-031-30620-4_7.

[38] G. Couteau, P. Rindal, and S. Raghuraman. "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes". In: *CRYPTO 2021*. Ed. by T. Malkin and C. Peikert. LNCS 12827. Springer, 2021, pp. 502–534. DOI: 10.1007/978-3-030-84252-9_17.

[39] G. Couteau and M. Zarezadeh. "Non-interactive Secure Computation of Inner-Product from LPN and LWE". In: *ASIACRYPT 2022*. Ed. by S. Agrawal and D. Lin. LNCS 13791. Springer, 2022, pp. 474–503. DOI: 10.1007/978-3-031-22963-3_16.

[40] A. Davidson, S. Katsumata, R. Nishimaki, S. Yamada, and T. Yamakawa. "Adaptively secure constrained pseudorandom functions in the standard model". In: *CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. LNCS 12170. Springer, 2020, pp. 559–589. DOI: 10.1007/978-3-030-56784-2_19.

[41] L. de Castro, C. Juvekar, and V. Vaikuntanathan. "Fast vector oblivious linear evaluation from ring learning with errors". In: *WAHC 2021: Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 2021, pp. 29–41. DOI: 10.1145/3474366.3486928.

[42] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[43] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. "Spooky encryption and its applications". In: *CRYPTO 2016*. Ed. by M. Robshaw and J. Katz. LNCS 9816. Springer, 2016, pp. 93–122. DOI: 10.1007/978-3-662-53015-3_4.

[44] P. Elias. "Error-correcting codes for list decoding". In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 5–12. DOI: 10.1109/18.61123.

[45] S. Garg, M. Mahmoody, D. Masny, and I. Meckler. "On the round complexity of OT extension". In: *CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. LNCS 10993. Springer, 2018, pp. 545–574. DOI: 10.1007/978-3-319-96878-0_19.

[46] N. Gilboa. "Two party RSA key generation". In: *CRYPTO 1999*. Ed. by M. J. Wiener. LNCS 1666. Springer, 1999, pp. 116–129. DOI: 10.1007/3-540-48405-1_8.

[47] O. Goldreich. "Candidate one-way functions based on expander graphs". In: *Studies in Complexity and Cryptography*. Ed. by O. Goldreich. LNCS 6650. Springer, 2011, pp. 76–87. DOI: 10.1007/978-3-642-22670-0_10.

[48] O. Goldreich, S. Micali, and A. Wigderson. "How to play any mental game, or a completeness theorem for protocols with honest majority". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Ed. by O. Goldreich. ACM, 2019, pp. 307–328. DOI: 10.1145/3335741.3335755.

[49] C. Guo, J. Katz, X. Wang, and Y. Yu. "Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers". In: *SP 2020*. IEEE, 2020, pp. 825–841. DOI: 10.1109/SP40000.2020.00016.

[50] J. HÅstad, R. Impagliazzo, L. A. Levin, and M. Luby. "A Pseudorandom Generator from any One-way Function". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396. DOI: 10.1137/S0097539793244708.

[51] R. Impagliazzo and S. Rudich. "Limits on the provable consequences of one-way permutations". In: *STOC 1989*. Ed. by D. S. Johnson. ACM, 1989, pp. 44–61. DOI: 10.1145/73007.73012.

[52] J. Indigo and D. Smart. *Page Weight: 2022: The Web Almanac by HTTP Archive*. 2022. URL: https://almanac.httparchive.org/en/2022/page-weight (visited on 02/29/2024).

[53] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. "Extending oblivious transfers efficiently". In: *CRYPTO 2003*. Ed. by D. Boneh. LNCS 2729. Springer, 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[54] Y. Ishai, M. Prabhakaran, and A. Sahai. "Secure arithmetic computation with no honest majority". In: *TCC 2009*. Ed. by O. Reingold. LNCS 5444. Springer, 2009, pp. 294–314. DOI: 10.1007/978-3-642-00457-5_18.

[55] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols*. 1st ed. Chapman and Hall/CRC, 2007. DOI: 10.1201/9781420010756.

[56] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. "Delegatable pseudorandom functions and applications". In: *CCS 2013*. Ed. by A. Sadeghi, V. D. Gligor, and M. Yung. ACM, 2013, pp. 669–684. DOI: 10.1145/2508859.2516668.

[57] J. Kilian. "Founding cryptography on oblivious transfer". In: *STOC 1988*. Ed. by J. Simon. ACM, 1988, pp. 20–31. DOI: 10.1145/62212.62215.

[58] V. Lyubashevsky, C. Peikert, and O. Regev. "A toolkit for ring-LWE cryptography". In: *EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. LNCS 7881. Springer, 2013, pp. 35–54. DOI: 10.1007/978-3-642-38348-9_3.

[59] V. Lyubashevsky, C. Peikert, and O. Regev. "On ideal lattices and learning with errors over rings". In: *EUROCRYPT 2010*. Ed. by H. Gilbert. LNCS 6110. Springer, 2010, pp. 1–23. DOI: 10.1007/978-3-642-13190-5_1.

[60] S. J. Menon and D. J. Wu. "SPIRAL: Fast, high-rate single-server PIR via FHE composition". In: *SP 2022*. IEEE, 2022, pp. 930–947. DOI: 10.1109/SP46214.2022.9833700.

[61] M. Naor and B. Pinkas. "Oblivious polynomial evaluation". In: *SIAM Journal on Computing* 35.5 (2006), pp. 1254–1281. DOI: 10.1137/S0097539704383633.

[62] M. Naor and O. Reingold. "Number-theoretic constructions of efficient pseudo-random functions". In: *Journal of the ACM* 51.2 (2004), pp. 231–262. DOI: 10.1145/972639.972643.

[63] C. Orlandi, P. Scholl, and S. Yakoubov. "The rise of Paillier: homomorphic secret sharing and public-key silent OT". In: *EUROCRYPT 2021*. Ed. by A. Canteaut and F. Standaert. LNCS 12696. Springer, 2021, pp. 678–708. DOI: 10.1007/978-3-030-77870-5_24.

[64] C. Peikert and S. Shiehian. "Privately constraining and programming PRFs, the LWE way". In: *PKC 2018*. Ed. by M. Abdalla and R. Dahab. LNCS 10770. Springer, 2018, pp. 675–701. DOI: 10.1007/978-3-319-76581-5_23.

[65] O. Peters. *Polymur universal hash function*. 2024. URL: https://github.com/orlp/polymur-hash (visited on 03/24/2024).

[66] K. Pietrzak and J. Sjödin. "Weak pseudorandom functions in minicrypt". In: *ICALP 2008*. Ed. by L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz. LNCS 5126. Springer, 2008, pp. 423–436. DOI: 10.1007/978-3-540-70583-3_35.

[67] S. Raghuraman, P. Rindal, and T. Tanguy. "Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN". In: *CRYPTO 2023*. Ed. by H. Handschuh and A. Lysyanskaya. LNCS 14084. Springer, 2023, pp. 602–632. DOI: 10.1007/978-3-031-38551-3_19.

[68] P. Rindal and L. Roy. *libOTe: an efficient, portable, and easy to use Oblivious Transfer Library*. URL: https://github.com/osu-crypto/libOTe (visited on 01/31/2024).

[69] L. Roy. "SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the Minicrypt model". In: *CRYPTO 2022*. Ed. by Y. Dodis and T. Shrimpton. LNCS 12507. Springer, 2022, pp. 657–687. DOI: 10.1007/978-3-031-15802-5_23.

[70] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. "Distributed Vector-OLE: Improved constructions and implementation". In: *CCS 2019*. Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM, 2019, pp. 1055–1072. DOI: 10.1145/3319535.3363228.

[71] S. Servan-Schreiber. *Constrained Pseudorandom Functions for Inner-Product Predicates from Weaker Assumptions*. 2024. Cryptology ePrint Archive, Report 2024/058.

[72] The OpenSSL Project. *OpenSSL: Cryptography and SSL/TLS Toolkit*. 2024. URL: https://www.openssl.org/ (visited on 02/12/2024).

[73] A. Ünal. *New Baselines for Local Pseudorandom Number Generators by Field Extensions*. 2023. Cryptology ePrint Archive, Report 2023/550.

[74] J. Yang, Q. Guo, T. Johansson, and M. Lentmaier. "Revisiting the concrete security of Goldreich's pseudorandom generator". In: *IEEE Transactions on Information Theory* 68.2 (2021), pp. 1329–1354. DOI: 10.1109/TIT.2021.3128315.

[75] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. "Ferret: Fast extension for correlated OT with small communication". In: *CCS 2020*. Ed. by J. Ligatti, X. Ou, J. Katz, and G. Vigna. ACM, 2020, pp. 1607–1626. DOI: 10.1145/3372297.3417276.

Supplementary Materials

A Additional Preliminaries

A.1 Oblivious Transfer

We define oblivious transfer functionality in Figure 15.

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

Parameters. String length k.

Parties. The functionality interacts with a sender S, receiver R.

Procedure.

1: Wait for input $(m_0, m_1) \in \{0,1\}^k$ from S.
2: Wait for input $b \in \{0,1\}$ from R.
3: Output m_b to the R and $\perp$ to S.

Fig. 15: Oblivious transfer ideal functionality $\mathcal{F}_{\mathsf{OT}}$.

A.2 RKA-secure PRFs

Here we define related-key attack (RKA)-secure PRFs.

Definition 10 (Φ-restricted Adversaries). *An efficient RKA-PRF adversary $\mathcal{A}$ is said to be Φ-restricted if its oracle queries use related-key derivation functions ϕ chosen arbitrarily from a set of valid key derivation functions Φ.*

Definition 11 (Related-Key-Attack Secure PRFs [16]). *Let $\lambda \in \mathbb{N}$ be a security parameter and $\ell = \ell(\lambda) \in \mathsf{poly}(\lambda)$. Let $\mathcal{F} = \{F_k : \mathcal{X}_\lambda \to \mathcal{Y}\}_{k \in \mathcal{K}_\lambda}$ be a family of functions and Φ: $\mathcal{K}_\lambda \to \mathcal{K}_\lambda$ be a family of related-key derivation functions. $\mathcal{F}$ is said to be an RKA-secure PRF family if for all efficient Φ-restricted adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},b}(\lambda)$ is negligible in λ. Here, b denotes the challenge bit and $\widetilde{\mathcal{F}}_\lambda$ denotes the set of all functions from $\Phi \times \mathcal{X}_\lambda$ to $\mathcal{Y}$.*

- ***Setup:*** *On input 1^λ, the challenger samples $k \xleftarrow{R} \mathcal{K}_\lambda$, and a uniformly random function $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, and runs $\mathcal{A}(1^\lambda)$.*
- ***Evaluation queries:*** *$\mathcal{A}$ adaptively sends arbitrary pairs $(\phi, x) \in \Phi \times \mathcal{X}_\lambda$ to the challenger. For each query (ϕ, x), the challenger proceeds as follows:*
  - *If $b = 0$, the challenger computes $y \leftarrow F_{\phi(k)}(x)$, and sends y to $\mathcal{A}$.*
  - *If $b = 1$, the challenger computes $y \leftarrow R(\phi, x)$ and sends y to $\mathcal{A}$.*
- ***Guess:*** *$\mathcal{A}$ outputs its guess b', which is the output of the experiment.*

$\mathcal{A}$ wins if $b' = b$, and its advantage $\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda)$ is defined as

$$\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) := \left| \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda) = 1] \right|,$$

where the probability is over the internal randomness of $\mathcal{A}$ and choice of k.

Definition 12 (Affine Related-Key Derivation Functions [1]). *Let $\mathcal{R}$ be a finite ring and let $m \geq 1$ be an integer, let the class Φ_{aff} (aff for affine) denote the class of functions from $\mathcal{R}^m$ to $\mathcal{R}^m$*

that can be separated into m component functions consisting of degree-1 univariate polynomials. That is,

$$\Phi_{\mathsf{aff}} := \left\{ \boldsymbol{\phi} : \mathcal{R}^m \to \mathcal{R}^m \mid \begin{array}{l} \boldsymbol{\phi} = (\phi_1, \ldots, \phi_m); \\ \forall i \in [m],\ \phi_i(k_i) = \gamma_i k_i + \delta_i,\ \gamma_i \neq 0 \end{array} \right\}.$$

Note that $\gamma_i \neq 0$ is necessary to make the derivation function non-trivial. The definition of Abdalla et al. [1] uses $\mathbb{Z}_p$; here we generalize it to any ring $\mathcal{R}$.

A.3 RKA-secure PRFs from random oracles

In this section, we include for completeness a proof of the (folklore) fact that when H is modeled as a random oracle, the function $F_k(x) = H(k, x)$ (for a suitable choice of the domain of k) is an RKA-secure PRF for affine relations.

Lemma 2. *Let $m = m(\lambda), n = n(\lambda), \ell(\lambda) \in \mathsf{poly}(\lambda)$ and let $\mathcal{R}$ be a ring. Let $H : \mathcal{R}^m \times \mathbb{F}_2^n \to \mathbb{F}_2^\ell$ be a hash function, modeled as a random oracle. Then, the family of functions $\mathcal{F} = \{x \mapsto H(k, x)\}_{k \in \mathcal{R}^m}$ is an RKA-secure PRF for the family Φ of all affine relations $\Phi = \{\phi_{\alpha,\beta} \colon k \mapsto \alpha k + \beta\}_{\alpha \in \mathcal{R}^*, \beta \in \mathcal{R}^m}$. More precisely, any Φ-restricted adversary $\mathcal{A}$ against the RKA security of $\mathcal{F}$, making at most q_e evaluation queries and q_r random oracle queries, has advantage at most*

$$\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) \leq \frac{q_r \cdot q_e}{\min_{\alpha \in \mathcal{R}^*} |\alpha \cdot \mathcal{R}^m|}.$$

Proof. Let Q_e denote the set of evaluation queries (ϕ, x) of $\mathcal{A}$, and let Q_r denote the set of random oracle queries of $\mathcal{A}$. Let us denote by Bad the following event during an execution of the experiment $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},b}(\lambda)$ (with $b = 0$ or $b = 1$): At the end of the experiment, the challenger computes $(\phi(k), x)$ for each $(\phi, x) \in Q_e$ and raises a flag Bad if $(\phi(k), x) \in Q_r$. If no flag Bad is raised, the challenger returns a flag Good.

Conditioned on the event Good, every answer $R(\phi, x)$ (where $R \colon \Phi \times \mathbb{F}_2^n \to \mathbb{F}_2^\ell$ is a random function) of the evaluation oracle to an evaluation query (ϕ, x) issued by $\mathcal{A}$ in $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda)$ is sampled as a fresh random element independent of $\mathcal{A}$'s view. In turn, it is distributed exactly as $H(\phi(k), x)$, because the latter is a fresh uniform random element when $\mathcal{A}$ never queries $(\phi(k), x)$—i.e., it is distributed exactly as in $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda)$. In other words,

$$\Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1 \mid \mathsf{Good}] = \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda) = 1 \mid \mathsf{Good}].$$

This implies

$$\begin{aligned} \mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) &= \left| \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1] \right| \\ &= \Pr[\mathsf{Bad}] \cdot \left| \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1 \mid \mathsf{Bad}] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda) = 1 \mid \mathsf{Bad}] \right| \\ &\leq \Pr[\mathsf{Bad}]. \end{aligned}$$

Now, fix a query $(\phi, x) \in Q_e$, and write $\phi \colon k \mapsto \alpha \cdot k + \beta$. Define $U := \{u \in \mathcal{R}^m \ : \ \exists v \in \mathbb{F}_2^n, (u, v) \in Q_r\}$. The probability, over the random choice of k, that $(\phi(k), x) \in Q_r$ is at most the probability that there exists $u \in U$ such that $\alpha \cdot k = u - \beta$. Since $\alpha \cdot k$ is a uniformly random element from the ideal $\alpha \cdot \mathcal{R}$, this happens with probability at most $q_r / |\alpha \mathcal{R}| \leq q_r / \min_{\alpha \in \mathcal{R}^*} |\alpha \mathcal{R}^m|$. The lemma then follows by a union bound over all queries in Q_e. ∎

B Application to large-scale MPC

We consider an application where a large number of parties are interacting over a network. Each individual is associated to a role (sender or receiver) and is identified by its public key. At any time, a pair of parties (S, R) with respective key pairs $(\mathsf{pk}_S, \mathsf{sk}_S)$ and $(\mathsf{pk}_R, \mathsf{sk}_R)$ can engage in a secure computation protocol, which reduces to a large number N of oblivious transfers [48]. Both parties derive PCF keys for ListOT via KeyDer, compute N ListOTs, and use them in N instances of the OT protocol from Figure 6. In this setting, we note that the ℓ-instance security notions of public key PCFs for ListOT captures the following security properties:

Sender security. Consider a sender S with key pair $(\mathsf{pk}_S, \mathsf{sk}_S)$ interacting with ℓ corrupted receivers $(R_1, \ldots, R_\ell)$ with public keys $(\mathsf{pk}_{R_1}, \ldots, \mathsf{pk}_{R_\ell})$. The sender has $N \cdot \ell$ message pairs $(m_0^{i,j}, m_1^{i,j})_{i \le N, j \le \ell}$. The sender and the receivers agree on $N \cdot \ell$ uniformly random inputs $(x_{i,j})_{i \le N, j \le \ell}$.

- **Key derivation.** The sender S sets $K_S^j \leftarrow \mathsf{pkPCF.KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_{R_j})$, for all $j \in [\ell]$.
- **Oblivious transfers.** For every $i \le N, j \le \ell$, S computes $(L_0^{i,j}, L_1^{i,j}) \leftarrow \mathsf{pkPCF.EvalS}(K_S^j, x_{i,j})$. For every $j \le \ell$, upon receiving $(c_{i,j})_{i \le N}$ from R_j, S replies with $(L_{c_{i,j}}^{i,j} \oplus m_0^{i,j}, L_{1-c_{i,j}}^{i,j} \oplus m_1^{i,j})_{i \le N}$ (following Figure 6).

To argue security in this setting, consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid is the protocol described above.

Hybrid $\mathcal{H}_1$. In this hybrid game, a simulator $\mathcal{S}$ (who is given the random tapes of the receivers $R_1, \cdots, R_\ell$, samples the lists $(L_0^{i,j}, L_1^{i,j})$ exactly as in $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,1}(\lambda, \ell(\lambda))$ from Figure 11, and plays the role of the sender exactly as in $\mathcal{H}_0$ afterwards. In this experiment, for any $i \le N$ and $j \le \ell$, the simulator $\mathcal{S}$ first samples $(L_0^{i,j}, L_1^{i,j})$ uniformly at random and then sets $L_{b_{i,j}}^{i,j}[\alpha_{i,j}] := v_{i,j}$, where $(b_{i,j}, v_{i,j}, \alpha_{i,j}) \leftarrow \mathsf{pkPCF.EvalR}(K_R^i, x_{i,j})$ (note that $\mathcal{S}$ can reconstruct the keys K_R^j from the tapes of the receivers). Importantly, for every (i,j), the list $L_{\bar{b}_{i,j}}^{i,j}$ remains truly random.

Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

Proof. This claim follows directly from the ℓ-instance sender security property of the public key PCF (cf. Figure 11). □

Hybrid $\mathcal{H}_2$. In this hybrid game, the simulator $\mathcal{S}$ first samples pk_S, reconstructs all keys K_R^j using pk_S by deriving sk_R^j from the tape of the receiver R_j, and computes $(b_{i,j}, v_{i,j}, \alpha_{i,j}) \leftarrow \mathsf{pkPCF.EvalR}(K_R^i, x_{i,j})$. For all $i \le N, j \le \ell$ it sends $b_{i,j}$ to the OT functionality, and obtains $m_{b_{i,j}}^{(i,j)}$. Finally, $\mathcal{S}$ samples $(L_0^{i,j}, L_1^{i,j})$ uniformly at random, and sets $L_{c_{i,j}}^{i,j}[\alpha_{i,j}] := v_{i,j} \oplus m_{b_{i,j}}^{(i,j)}$. For $j = 1$ to ℓ, $\mathcal{S}$ sends $(L_{c_{i,j}}^{i,j}, L_{\bar{c}_{i,j}}^{i,j})_{i \le N}$ to R_j.

Claim. $\mathcal{H}_2 \approx \mathcal{H}_1$.

Proof. Note that the change in $\mathcal{H}_2$ is a purely syntactic, because all values in $L_{\bar{c}_{i,j}}$ and all values $L_{c_{i,j}}[k]$ for $k \neq \alpha_{i,j}$ are uniformly and independently random.

This concludes the proof of sender security.

Receiver security. Conversely, consider a receiver R with key pair $(\mathsf{pk}_R, \mathsf{sk}_R)$ interacting with ℓ corrupted senders $(S_1, \cdots, S_\ell)$ with public keys $(\mathsf{pk}_{S_1}, \cdots, \mathsf{pk}_{S_\ell})$. The receiver has $N \cdot \ell$ selection bits $(b_{i,j})_{i \le N, j \le \ell}$. The senders and the receiver agree on $N \cdot \ell$ uniformly random inputs $(x_{i,j})_{i \le N, j \le \ell}$.

- **Key derivation.** R sets $K_R^j \leftarrow \mathsf{pkPCF.KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_{S_j})$ for $j \in [\ell]$.
- **Oblivious transfers.** For every $i \le N, j \le \ell$, R computes $(b'_{i,j}, v_{i,j}, \alpha_{i,j}) \leftarrow \mathsf{pkPCF.EvalR}(K_R^j, x_{i,j})$. For every $j \le \ell$, the receiver sends $c_{i,j} \leftarrow b_{i,j} \oplus b'_{i,j}$ to S_j. Upon receiving $(L_0^{i,j}, L_1^{i,j})_{i \le N}$ from S_j, R outputs $m_{i,j} \leftarrow L_{b_{i,j}}^{i,j}[\alpha_{i,j}] \oplus v_{i,j}$.

The security analysis is straightforward: The simulator samples the bits $b_{i,j}$ uniformly at random, as in the experiment $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,1}(\lambda, \ell(\lambda))$ from Figure 12. The simulated game is indistinguishable from the honest game by the ℓ-instance receiver security of pkPCF. In this simulated game, $c_{i,j} = b_{i,j} \oplus b'_{i,j}$ perfectly masks $b_{i,j}$, and receiver security follows.

C Precomputability, Two-Round OT Extension, and More

In this section, we discuss additional features offered by our framework and instantiations. In particular, we describe how our framework offers precomputability for either the sender or the receiver, with an efficient "synchronization" protocol using standard building blocks. We additionally discuss several other features offered by our framework, such as two-round OT extension and theoretical instantiations in the standard model.

Tool: Vector Oblivious Linear Evaluation (VOLE). As a building-block for precomputability and two-round OT extension, we first describe Vector OLE [61, 54] and "reverse" VOLE (reVOLE) [24]. We define the ideal functionalities for these primitives in Figure 16, when generalized to matrix inputs (this generalization follows immediately from standard VOLE and reVOLE). At a high level, matrix-VOLE allows the receiver to obtain $\mathbf{A}+\mathbf{b}\mathbf{x}^\top$ as output (the sender gets no output), when the sender inputs $(\mathbf{A}, \mathbf{b})$ and the receiver inputs $\mathbf{x}$. In contrast, matrix-reVOLE allows the receiver to obtain $\mathbf{A}+\mathbf{b}\mathbf{x}^\top$ as output, when the sender inputs $(\mathbf{A}, \mathbf{x})$ and the receiver inputs $\mathbf{b}$. Using any matrix-VOLE (or matrix-reVOLE) protocol, we obtain precomputability for the sender and receiver via simple building blocks (VOLE and reVOLE).

Parameters. Finite ring $\mathcal{R}$ and integers $m, n \geq 1$.

Parties. The functionalities interact with a sender S and a receiver R.

Ideal functionality $\mathcal{F}_{\mathsf{VOLE}}$:

1: Wait for input $(\mathbf{A}, \mathbf{b}) \in \mathcal{R}^{m\times n} \times \mathcal{R}^m$ from S.
2: Wait for input $\mathbf{x} \in \mathcal{R}^n$ from R.
3: Compute $\mathbf{C} \leftarrow \mathbf{A} + \mathbf{b}\mathbf{x}^\top$, and output $\mathbf{C}$ to R and $\perp$ to S.

Ideal functionality $\mathcal{F}_{\mathsf{reVOLE}}$:

1: Wait for input $(\mathbf{A}, \mathbf{x}) \in \mathcal{R}^{m\times n} \times \mathcal{R}^n$ from S.
2: Wait for input $\mathbf{b} \in \mathcal{R}^m$ from R.
3: Compute $\mathbf{C} \leftarrow \mathbf{A} + \mathbf{b}\mathbf{x}^\top$, and output $\mathbf{C}$ to R and $\perp$ to S.

Fig. 16: Ideal functionalities for matrix-VOLE ($\mathcal{F}_{\mathsf{VOLE}}$) [61, 54].

C.1 Precomputability

Here, we describe how either the sender or receiver can precompute all their inputs ahead of time, *without needing to know the identity of the other party.* Our definition for precomputability is inspired by the blueprint laid out by Couteau, Meyer, Passelègue, and Riahinia [37].[17] Moreover, we explain how VOLE or (reVOLE) can be used to efficiently synchronize the parties after one of the parties precomputes their input.

Remark 4. The definition of precomputability in [37, Def. 7] has downsides which we address here in Definition 13. In particular, their definition does not rule out a "trivially precomputable" scheme where KeyGen_0 outputs both the keys (and the other party's key would be fixed by having it as part of the auxiliary information passed to KeyGen_1). In this case the first party has both keys and we do not get meaningful security for the applications when using this setup. Additionally, their definition provides no formalization of the interactive "synchronization" protocol, which we resolve by defining the ideal functionality used by the parties to agree on common keys (without changing the first party's key) following one party's precomputation.

Definition 13 (Precomputable PCF for ListOT). *Let λ be a security parameter and $\lambda \leq n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. Let $\mathsf{PCF} = (\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ be a PCF for ListOT. We say that PCF is* sender precomputable *if there are algorithms KeyGen_S, $\mathsf{KeyGenRev}_S$, and an interactive protocol KG_R between the sender S and a receiver R, such that:*

[17] Note that *simultaneous* precomputability is not possible [37].

– $\mathsf{KeyGen}_S(1^\lambda) \to K_S$. *Takes as input the security parameter. Outputs a key* K_S.
– $\mathsf{KeyGenRev}_S(K_S) \to r$. *Takes as input a sender's key, and outputs the sampling randomness* $r \in \{0,1\}^*$. *(This is used to formalize a notion of oblivous sampleability of the key.)*
– $\mathsf{KG}_R\langle S, R\rangle$ *is an interactive protocol, where* S *has input* K_S *(as output by the first algorithm), and* R *has no input. After the protocol,* R *outputs a key* K_R*, while* S *outputs* $\perp$.

Importantly, we have that the keys generated by this process are identically distributed to those generated by $\mathsf{PCF.KeyGen}$. *That is, we first require that the sender's key is* obliviously sampleable *(independently of the receiver's key), a notion that is inspired by [32]. Formally,*

1. *Generating the sender's key via* KeyGen_S *is perfectly indistinguishable from the generating it via* $\mathsf{PCF.KeyGen}$*:*

$$\left\{K_S \mid K_S \leftarrow \mathsf{KeyGen}_S(1^\lambda)\right\} \equiv \left\{K_S \mid (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda)\right\},$$

   and
2. *the algorithm* $\mathsf{KeyGenRev}_S$ *returns, for a sender's key* K_S*, the randomness that is used to generate it using* KeyGen_S*:*

$$\left\{(K_S, r) \mid r \xleftarrow{R} \{0,1\}^{\mathsf{poly}(\lambda)}; K_S := \mathsf{KeyGen}_S(1^\lambda; r)\right\} \\ \approx_c \left\{(K_S, r) \mid (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda); r \leftarrow \mathsf{KeyGenRev}_S(K_S);\right\}.$$

Additionally, we require that KG_R *securely realizes the functionality* $\mathcal{F}_{\mathsf{KG},R}$ *given in Figure 17.*

Receiver precomputability. *This is defined in the exact analogous way, where all instances of* S *and* R *are exchanged. Hence, it asks for the existence of* KeyGen_R*,* $\mathsf{KeyGenRev}_R$*, and an interactive protocol* KG_S *that fulfil the respective analogous properties.*

Parties. The functionalities interacts with a sender S and a receiver R.

Ideal functionality $\mathcal{F}_{\mathsf{KG},R}$:

1: Wait for input K_S from S.
2: Wait for input ("`KeyGen`") from R.
3: Compute $K_R \leftarrow \mathsf{KeyDer}_R(K_S)$, where KeyDer_R is an algorithm that computes a fitting receiver's key (i.e., such that the distribution of (K_S, K_R) is identical to the output of $\mathsf{PCF.KeyGen}(1^\lambda)$) from a valid sender's key K_S. Then, output K_R to R and $\perp$ to S.

Fig. 17: Ideal functionality for the protocol that securely establishes a fitting key for the other party, after one party's key has been generated ahead of time.

The main motivation of precomputability is that one party can locally generate a key and use it to precompute all evaluations. Then, at a later point in time, another party can use an interactive protocol that securely establishes a key for that "fits" to the key that was used by the first party for the precomputation. We now turn to describing how we can achieve precomputability for either the sender or the receiver.

Sender precomputability. KeyGen_S simply outputs the ShCPRF master key, exactly as in the construction. This gives us sender precomputability for Section 5, because in the construction, we have $K_S = \mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0})$ (a ShCPRF master key as defined in Figure 1) which is sampled independently of the receiver's key K_R. Then, to obtain K_R, the receiver can invoke a matrix-VOLE protocol, defined in Figure 16, where the sender inputs $(\mathbf{Z_0}, \Delta)$ while the receiver inputs $-\mathbf{z}$ (note that $\Delta \in \mathcal{R}^m$ and $\mathbf{z} \in \mathcal{R}^n$). The receiver then obtains $\mathbf{Z_1} = \mathbf{Z_0} - \Delta\mathbf{z}^\top$ as output, which is distributed identically to $\mathbf{Z_1}$ in Figure 1. Note that $\mathbf{k_0}$, which is common to both msk and csk, can simply be sent over by the sender. As such, the receiver successfully derives $K_R = ((\mathbf{k_0}, \mathbf{Z_1}), \mathbf{z})$, which matches the expected receiver key.

Receiver precomputability. Showing that we also get *receiver* precomputability is slightly more involved. First, note that we can let the receiver sample $\mathbf{k_0}$, $\mathbf{Z_1}$, and $\mathbf{z}$ uniformly which together form $K_R = ((\mathbf{k_0}, \mathbf{Z_1}), \mathbf{z})$. Therefore, we can define KeyGen_R to output these elements, which are distributed identically to the receiver's key in the construction. Using K_R, the receiver can precompute all the OT messages for Figure 6.

Then, the challenge is finding a way for the *sender* (who has Δ) to obtain the "master key" $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0})$ where $\mathbf{Z_0} = \mathbf{Z_1} + \Delta\mathbf{z}^\top$. This can be achieved by invoking a matrix-reVOLE protocol, defined in Figure 16, where now the receiver (playing the role of the sender) inputs $(\mathbf{Z_1}, \mathbf{z})$ and the sender (playing the role of the receiver) inputs $\Delta \in \mathcal{R}^m$. The sender obtains as output $\mathbf{Z_0} = \mathbf{Z_1} + \Delta\mathbf{z}^\top$, which is distributed identically to $\mathbf{Z_0}$ in Figure 1. Once more, the receiver can simply send $\mathbf{k_0}$ to the sender. This allows the sender to recover $K_S = (\mathbf{k_0}, \mathbf{Z_0})$, as required.

Application: Fast OTs in a client-server model. To motivate the notion of precomputability, consider the following setting: a weak client will want to, at some point in the future, run a secure two-party computation protocol with some servers, the identity of which is yet unknown. During an idle period, the weak client generates its key $(K_0, \mathsf{aux}) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$, and precomputes a large number of ListOTs (either list pairs (L_0, L_1) if the client has a sender role, or triples (b, v, α)). When the client decides on a server they want to generate OTs with, a cheap distributed protocol is performed (by our above discussion, this can be done with a single length-n VOLE or reVOLE with our construction) to provide the key K_1 to the server. The powerful server can then quickly compute its share of the ListOTs on-the-fly, and engage in the two-party computation with the client. In a typical scenario, the client could be someone's phone; during night, the phone could automatically prepare ListOTs. Then, whenever the phone holder decides to browse a website, the phone could enable a two-party computation with the website (e.g., to securely get restaurant recommendations, find matching profiles, or be served with a targeted ad) using only cheap computations on its side (indeed, the bottleneck becomes entirely communication).

C.2 Two-round setup from two-round OT

In this section, we describe how QuietOT yields a protocol for two-round OT extension. Since two-round OT is clearly optimal, two-round OT extension provides the best possible performance one could hope for (in terms of rounds and use of symmetric vs. public-key primitives). However, due to the impossibility result of Garg et al. [45], two-round OT extension is impossible to instantiate unconditionally in the ROM, and therefore *necessitates* a non-black-box assumption. Here, we show that using any black-box two-round OT protocol, we get a two-round key-derivation protocol for our PCF for ListOT from Figure 5. This then allows us to build a two-round OT extension. Coupled with our instantiation of QuietOT in the standard model (see Appendix C.3), we show that any IPM-wPRF suffices to achieve two-round OT extension and circumvent the impossibility of Garg et al. [45] using a broad family of Minicrypt primitives.

Lemma 3 (VOLE and reVOLE from OT). *Two-round OT implies two-round VOLE and reVOLE, as defined in Figure 16.*

Proof. Two round reverse VOLE can be constructed using n parallel calls to a two-round OLE functionality (where the receiver inputs $\mathbf{x} \in \mathcal{R}^n$ coordinate-by-coordinate and the sender inputs $\mathbf{A}$ column-by-column and $\mathbf{b}$). In turn, two-round OLE can be constructed from $O(m \log_2 |\mathcal{R}|)$ parallel calls to a two-round OT functionality using the protocol of Gilboa [46] for ring $\mathcal{R}^m$. The same holds for reVOLE, since reVOLE is trivially implied by OLE [8, 3]. ■

Theorem 6 (Informal). *There exists a two-round key derivation protocol for Figure 5 from any two-round OT protocol.*

Proof. The proof follows from the protocols described in Appendix C.1 and using Lemma 3. Consider the PCF for ListOT framework of Figure 5. As shown in Appendix C.1, using a reVOLE protocol, the receiver can obtain K_R with one call to the reVOLE functionality by inputting $-\mathbf{z}$ and having the sender input $(\mathbf{Z_0}, \Delta)$. Again, since $\mathbf{k_0}$ is common to both the sender and receiver in Figure 5, it can be transmitted separately. By Lemma 3, this reVOLE functionality can be instantiated using $O(n \cdot m \log_2(|\mathcal{R}|))$ parallel calls to a two-round OT functionality, which implies that the receiver can derive K_R in two rounds. ■

C.2.1 Two-round OT extension. Note that, just using $\mathbf{z}$ (the wPRF key) and the random x_i, the receiver can precompute all the OT messages sent to the sender in Figure 6, without needing to know csk (recall that $K_R = (\mathsf{csk}, \mathbf{z})$). Specifically, by (1) computing all the bits destined for the sender in Figure 6 using $\mathbf{z}$, and (2) executing the two-round key-derivation protocol from Theorem 6 in parallel with Figure 6, the receiver obtains csk and all the information it needs to decode the response messages received from the sender. A little more concretely,

Round 1: Receiver → Sender. In the first round, the receiver, with choice bits $\mathbf{b} = (b_1, \ldots, b_N)$, starts by computing all the bit masks $\mathbf{b}'$ using the wPRF key $\mathbf{z}$ and x and sends $\mathbf{c} = \mathbf{b} \oplus \mathbf{b}'$ to the sender in Figure 6. In addition, the receiver also sends the reVOLE message used to derive csk in parallel with its choice bits.

Round 2: Sender → Receiver. The sender computes the lists L_0 and L_1 as in Figure 6, then using the masked bits $(c_i \in \mathbf{c})_{i\in[N]}$ received from the receiver, it responds with its reVOLE response message and lists $(L'_{i,0}, L'_{i,1})_{i\in[N]}$. The receiver can then locally (1) reconstruct csk from the reVOLE response message and (2) recover the messages exactly as in Figure 6.

C.3 Instantiations of QuietOT in the standard model

An interesting feature of our PCF for ListOT framework (Figure 5) is that security (for the sender) reduces entirely to the ShCPRF, as seen in the proof of Theorem 2. While the simplest instantiation of the ShCPRF framework (Section 5) is using a random oracle, *any* suitable RKA-secure PRF suffices. In particular, the work of Servan-Schreiber [71] which shows that the VDLPN wPRF candidate of Boyle et al. [23] can be used to instantiate a (weak) CPRF (in turn giving us a weak ShCPRF by extension). Coupled with the work of Bui et al. [30] which shows that the VDLPN wPRF candidate is actually an IPM-wPRF, we can instantiate QuietOT solely based on the VDLPN assumption. Of course, using alternative CPRF constructions supporting inner-product predicates based on DDH [71], DCR [37], or LWE [40] is also an option. However, while such instantiations are interesting when viewed from a theoretical lens, they do not lead to practical constructions given their "public-key" nature.

Remark 5. We note that Applebaum, Harnik, and Ishai [4] have shown that it is possible to instantiate the IKNP OT extension protocol assuming RKA-secure PRFs, which coupled with our framework, makes studying the relationship between RKA-security and OT extension an interesting direction for future work.

Remark 6 (Generating random inputs). In practice, the inputs to the PCF (which are used as inputs to the IPM-wPRF after map) need to be uniformly random. The random oracle model immediately implies a common random string available to both parties to use as inputs. However, if we replace the random oracle with an RKA-secure PRF, then the parties need a different way to obtain uniformly random inputs. One idea is to settle for *pseudorandom* inputs and have both parties obtain a common seed for a PRG (or alternatively obtain a PRF key), which they can expand into many (pseudo)random inputs $x_1, \ldots, x_N$ to generate N correlations. Unfortunately, such an approach is only heuristically secure in the general case, since there exist counter-examples to the security of wPRFs when evaluated using (public) pseudorandomness [66]. However, for the BIPSW and VDLPN wPRF candidates, security is believed to hold even when evaluated using public pseudorandomness, as shown in a recent work of Brzuska et al. [29].

D Deferred proofs

D.1 Proof of Theorem 1

Proof. We prove correctness, security, and pseudorandomness in turn.

Correctness. Consider a constraint $\mathbf{z} \in \mathcal{R}^n$ and input $\mathbf{x} \in \mathcal{R}^n$ such that $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e, the constraint is satisfied). It holds that $\mathbf{k} = \mathbf{k}_0 + \mathbf{Z_0 x} = \mathbf{k}_0 + \mathbf{Z_1 x} + (\Delta \mathbf{z}^\top)\mathbf{x} = \mathbf{k}_0 + \mathbf{Z_1 x}$. Therefore, the resulting $\mathbf{k}$ is identical in Eval and CEval of Figure 1 for the same input $\mathbf{x}$. Correctness then follows from the correctness of F. For shiftability correctness, for all $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha \neq 0$, using shift α, then $\mathbf{k} = \mathbf{Z_0 x} - \Delta\alpha = \mathbf{k} = \mathbf{Z_0 x} - \Delta\alpha + \Delta \langle \mathbf{z}, x \rangle - \Delta \langle \mathbf{z}, \mathbf{x} \rangle = \mathbf{Z_1 x} + \Delta \langle \mathbf{z}, \mathbf{x} \rangle - \Delta\alpha$. Therefore,

when $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$, the resulting $\mathbf{k}$ is identical in Eval and CEval of Figure 1 for the same input $\mathbf{x}$ and correctness follows.

(1-key, selective) Security. We prove security by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

Hybrid $\mathcal{H}_0$. This hybrid consists of the (1-key, selective) ShCPRF security game defined in Definition 3.

Hybrid $\mathcal{H}_1$. In this hybrid, during setup, the challenger first samples the constrained key and *then* samples the master key. Specifically, at the start of the game, given the constraint $\mathbf{z} \in \mathcal{R}^n$, the challenger first samples a constrained key $\mathsf{csk} := (\mathbf{k_0}, \mathbf{Z_1})$, where $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m$ and $\mathbf{Z_1} \xleftarrow{R} \mathcal{R}^{m \times n}$. Then, the challenger computes the master secret key as $\mathsf{msk} := (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$, where $\Delta \xleftarrow{R} \mathcal{R}^m$, $\mathbf{Z_0} := \mathbf{Z_1} + \Delta \mathbf{z}^\top$ and $\mathbf{k}_0$ is as in csk. The difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is purely syntactic. In particular, it follows that the distribution of msk and csk in $\mathcal{H}_1$ is identical to $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In this hybrid game, the challenger does not sample Δ anymore. Instead, the challenger is given access to the following stateful oracle $\mathcal{O}_{\mathsf{rka}}$:

Oracle $\mathcal{O}_{\mathsf{rka}}$

Initialize. Sample $\Delta \xleftarrow{R} \mathcal{R}^m$.

Evaluation. On input $\phi \in \Phi_{\mathsf{aff}}$ and $\mathbf{x} \in \mathcal{R}^n$, return $F_{\phi(\Delta)}(\mathbf{x})$.

The challenger is then defined as follows.

1. **Setup:** On input 1^λ, the challenger
   - runs $\mathcal{A}(1^\lambda)$ who outputs a constraint $\mathbf{z}$;
   - samples csk according to $\mathcal{H}_1$ by sampling $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m$, $\mathbf{Z_1} \xleftarrow{R} \mathcal{R}^{m \times n}$;
   - samples a uniformly random function $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, where $\widetilde{\mathcal{F}}_\lambda$ is the set of all functions with domain $\mathcal{X} \times \mathcal{S}$ and range $\mathcal{Y}$; and
   - sends csk to $\mathcal{A}$.
2. **Evaluation queries:** For each query $(\mathbf{x}, \alpha)$ from $\mathcal{A}$, if $C_{\mathbf{z}}(x, \alpha) = 0$, then the challenger responds with $\perp$. Otherwise, the challenger proceeds as follows:
   - If $b = 0$, it computes $a := \langle \mathbf{z}, \mathbf{x} \rangle - \alpha$ and $\mathbf{b} := \mathbf{k_0} + \mathbf{Z_1}\mathbf{x}$, defines the affine function $\phi \colon \mathbf{u} \mapsto a\mathbf{u} + \mathbf{b}$, queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi, \mathbf{x})$, and forwards the response y to $\mathcal{A}$.
     ▷ We note that y is computed by $\mathcal{O}_{\mathsf{rka}}$ as $F_{\mathbf{k}'}(\mathbf{x})$ where
     ▷ $\mathbf{k}' = a\Delta + \mathbf{b} \in \mathcal{R}^m = (\langle \mathbf{z}, \mathbf{x} \rangle - \alpha)\Delta + \mathbf{b} = \phi(\Delta)$, for some $\phi \in \Phi_{\mathsf{aff}}$.
   - If $b = 1$, it computes $y \leftarrow R(x, \alpha)$ and returns y,

Claim. $\mathcal{A}$'s advantage in $\mathcal{H}_2$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_1$.

Proof. The difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is again purely syntactic since each output is computed identically in both games. However, note that the challenger now only has access to Δ via the oracle $\mathcal{O}_{\mathsf{rka}}$. □

Claim. There does not exist an efficient $\mathcal{A}$ with greater than negligible advantage in $\mathcal{H}_2$ assuming $\mathcal{F}$ is an RKA-secure PRF with respect to affine related key derivation functions Φ_{aff}.

Proof. Note that the challenger in $\mathcal{H}_2$ is already playing the role of a Φ_{aff}-restricted adversary when querying the oracle $\mathcal{O}_{\mathsf{rka}}$ to answer the evaluation queries. The reduction to RKA security of $\mathcal{F}$ is therefore immediate. □

This concludes the proof of (1-key, selective) security.

Pseudorandomness. We prove the pseudorandomness property by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

Hybrid $\mathcal{H}_0$. In this hybrid, the adversary is given oracle access to $\mathsf{Eval}(\mathsf{msk}, \cdot, \cdot)$.

Hybrid $\mathcal{H}_1$. In this hybrid, the reduction emulates the answers of the oracle queries of $\mathcal{A}$ as follows. It samples $\mathbf{Z}_0 \xleftarrow{R} \mathcal{R}^{m \times n}$ and $\Delta \xleftarrow{R} \mathcal{R}^m \setminus \{0\}$. In addition, the reduction interacts with the following oracle $\mathcal{O}_{\mathsf{rka}}$:

Oracle $\mathcal{O}_{\mathsf{rka}}$

Initialize. Sample $\mathbf{k}_0 \xleftarrow{R} \mathcal{R}^m$.

Evaluation. On input $\phi \in \Phi_{\mathsf{aff}}$ and $\mathbf{x} \in \mathcal{R}^n$, return $F_{\phi(\mathbf{k}_0)}(\mathbf{x})$.

Given a query $(\mathbf{x}, \alpha)$, the reduction defines $\phi_\alpha : \mathbf{k} \mapsto \mathbf{k} + \mathbf{Z}_0\mathbf{x} - \Delta \cdot \alpha$, and queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi_\alpha, \mathbf{x})$, and forwards the response to $\mathcal{A}$. Observe that the answers to $\mathcal{A}$'s queries in $\mathcal{H}_1$ are always equal to $\mathsf{Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$ for $\mathsf{msk} := (\mathbf{k}_0, \mathbf{Z}_0, \Delta)$, hence $\mathcal{A}$'s advantage in $\mathcal{H}_1$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In this hybrid, the answer of $\mathcal{O}_{\mathsf{rka}}$ on a query $(\mathbf{x}, \alpha)$ is computed as $R(\phi_\alpha, \mathbf{x})$, where R is a uniformly random function from the set $\widetilde{\mathcal{F}}_\lambda$ of all functions from $\Phi_{\mathsf{aff}} \times \mathcal{S}$ to $\mathcal{Y}$. By the RKA-security of the PRF family, $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable.

Hybrid $\mathcal{H}_3$. In this hybrid, we sample a uniformly random function $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, where $\widetilde{\mathcal{F}}_\lambda$ is the set of all functions from $\mathcal{R}^n \times \mathcal{S}$ to $\mathcal{Y}$, and all queries of $\mathcal{A}$ are answered with R. Observe that for any two queries $(\mathbf{x}_0, \alpha_0)$ and $(\mathbf{x}_1, \alpha_1)$, it holds that $(\phi_{\alpha_0}, \mathbf{x}_0) = (\phi_{\alpha_1}, \mathbf{x}_1)$ iff $(\mathbf{x}_0, \alpha_0) = (\mathbf{x}_1, \alpha_1)$, hence $\mathcal{H}_3$ is perfectly indistinguishable from $\mathcal{H}_2$.

This concludes the proof of pseudorandomness and the proof of Theorem 1. ■

D.2 Proof of Theorem 2

Proof. We prove pseudorandomness, correctness, sender security, and receiver security in turn.

Pseudorandomness. We prove pseudorandomness via two hybrid games.

Hybrid $\mathcal{H}_0$. This hybrid consists of $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$ from Figure 2, where $\mathsf{PCF.KeyGen}$ and $\mathsf{PCF.Eval}$ are as defined in Figure 5 (PCF for ListOT framework).

Hybrid $\mathcal{H}_1$ (wPRF security). In this hybrid game, we replace each pseudorandom bit b_i computed in $\mathcal{H}_0$ using the IPM-wPRF f, with truly random bits.

Claim. $\mathcal{H}_0 \approx_c \mathcal{H}_1$.

Proof. Suppose, towards a contradiction, that there exists an efficient $\mathcal{A}$ that distinguishes between $\mathcal{H}_0$ and $\mathcal{H}_1$ with non-negligible advantage $\nu(\lambda)$. Since the only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that the pseudorandom bits in $\mathcal{H}_0$ are replaced with uniformly random bits in $\mathcal{H}_1$, the reduction to the wPRF pseudorandomness of f is immediate. (For this, note that $\mathbf{z}$ is independent of msk.) □

Hybrid $\mathcal{H}_2$ (ShCPRF pseudorandomness). In this hybrid, for all $i \in [N(\lambda)]$, the lists L_0^i, L_1^i are sampled uniformly from $\mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_0)$ and $\mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_1)$, respectively, where the distribution $\mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(\cdot)$ is defined in Definition 5 and consists of uniformly random samples from $\mathcal{Y}$.

Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$.

Proof. Suppose, towards a contradiction, that there exists an efficient $\mathcal{A}$ that distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$ with non-negligible advantage $\nu(\lambda)$. We can then construct an efficient $\mathcal{B}$ that contradicts the pseudorandomness property of the ShCPRF (Definition 3). Note that in Figure 5, the list entries of L_0^i, L_1^i are sampled as

$$s_0^i \leftarrow \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_0) \text{ and } s_1^i \leftarrow \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_1),$$

for all $\alpha_0 \in S_0$ and $\alpha_1 \in S_1$, where $\mathbf{x}_i \leftarrow \mathsf{map}(x_i) \in \mathcal{R}^n$, and then assembled into the two lists L_0^i, L_1^i.

Given oracle access to $\mathcal{O}$ which is either a random function $R(\cdot, \cdot)$ or the algorithm $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \cdot, \cdot)$, we construct $\mathcal{B}$ as follows:

1. For all $i \in [N(\lambda)]$,
   - sample $x_i \xleftarrow{R} \mathcal{X}_\lambda$ and set $\mathbf{x}_i \leftarrow \mathsf{map}(x_i) \in \mathcal{R}^n$,
   - query $(\mathbf{x}_i, \alpha_0)$ to the oracle on all $\alpha_0 \in S_0$ to get response s_{α_0},
   - query $(\mathbf{x}_i, \alpha_1)$ to the oracle on all $\alpha_1 \in S_1$ to get response s_{α_1},
   - sample bit b_i uniformly at random.
2. Then, assemble the lists L_0^i, L_1^i and run $\mathcal{A}$ on input $(1^\lambda, (x_i, L_0^i, L_1^i, b_i)_{i\in[N(\lambda)]})$ and output as it does.

Observe that the lists $(L_0^i, L_1^i)_{i\in[N(\lambda)]}$ output by $\mathcal{B}$ are distributed identically to the lists in Figure 5 if $\mathcal{B}$ is given oracle access to the ShCPRF, and distributed as uniformly random lists when $\mathcal{B}$ is given oracle access to a random function. Therefore, the distribution given to $\mathcal{A}$ is identical to $\mathcal{H}_1$ or $\mathcal{H}_2$, allowing $\mathcal{B}$ to win the pseudorandomness game of the ShCPRF with the same advantage. □

We observe that $\mathcal{H}_2$ is already identical to $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$, which concludes the proof of pseudorandomness.

Correctness. Observe that v output by $\mathsf{PCF.EvalR}(K_R, x)$ is computed as $v \leftarrow \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$, where $\mathbf{x} \leftarrow \mathsf{map}(x)$ and $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$ for a random constraint $\mathbf{z}$. Hence, by correctness of the ShCPRF, we know that there exists a shift $\alpha \in S_0 \cup S_1$, such that $\mathbf{x}$ and α are authorized. More specifically, $C(\mathbf{x}, \alpha) := \langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$, and with overwhelming probability, v is equal to the list entry calculated via $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$. (Note that α is calculated in $\mathsf{PCF.EvalR}$ exactly in this way.) Hence, for the (unique) $b' \in \{0, 1\}$ with $\alpha \in S_{b'}$, we have that, with overwhelming probability, $L_{b'}[\alpha] = v$. Finally, $b' = b$ by the property of the IPM-wPRF which guarantees that $b := f_{\mathbf{z}}(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_0$ and $f_{\mathbf{z}}(\mathbf{x}) = 1$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_1$.

Sender Security. Informally, this follows from the fact that by the shiftable CPRF security, for any (possibly not random)[18] $x_i \xleftarrow{R} \mathcal{X}$ and given only csk for a constraint $\mathbf{z}$, all constrained values are pseudorandom to the adversary. However, for the given $\mathbf{x}_i \leftarrow \mathsf{map}(x_i)$, it is authorized only exactly for this constraint $\mathbf{z}$ with shift α_i (because $S_0 \cup S_1 = \mathcal{R}$ and $\alpha_i = \langle \mathbf{z}, \mathbf{x} \rangle \in \mathcal{R}$ is the unique value such that $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha_i = 0$). More formally, we have a sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda)$ experiment defined in Figure 3, where $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{EvalS}, \mathsf{EvalR})$ are as defined in Figure 5 (PCF for ListOT framework). In particular, we note that $\mathsf{PCF.EvalS}$ internally runs the ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$.

Hybrid $\mathcal{H}_1$. In this hybrid, for each $i \in [N(\lambda)]$, the call to $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$ inside of $\mathsf{PCF.EvalS}$ is replaced with a call to $\mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$ whenever $(\mathbf{x}, \alpha)$ is authorized (i.e., $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$), where csk is part of K_R in $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda)$.

Claim. $\mathcal{H}_0 \approx_s \mathcal{H}_1$.

Proof. By the correctness of shiftable CPRFs, we have that for all authorized $(\mathbf{x}_i, \alpha_i)$ pairs,

$$\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_i) = \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x}_i),$$

with overwhelming probability. □

Hybrid $\mathcal{H}_2$. In this hybrid, the lists L_0^i, L_1^i are sampled uniformly at random from $\mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_0), \mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_1)$. Then, for each i where $(\mathbf{x}_i, \alpha_i)$ is an authorized pair, find the $b_i \in \{0, 1\}$, such that $\alpha_i \in S_{b_i}$, and overwrite $L_{b_i}[\alpha_i] \leftarrow \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x}_i)$.

Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_2$.

Proof. The claim follows directly from from the security of the shiftable CPRF. Namely, let $\mathcal{A}$ be an efficient adversary with a non-negligible advantage of distinguishing between $\mathcal{H}_1$ and $\mathcal{H}_2$. Then, we define the following adversary $\mathcal{B}$ to the (1-key, selective) security experiment $\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{B}}(\lambda)$ for ShCPRF. When $\mathcal{B}$ is queried for a constraint, it samples a random $\mathbf{z} \xleftarrow{R} \mathcal{R}^n$ (or via some distribution over $\mathcal{R}^n$) and outputs it. When $\mathcal{B}$ is then run with a key csk, it sets $K_R := (\mathsf{csk}, \mathbf{z})$. Then, for each $i \in [N(\lambda)]$,

[18] E.g., in the GAR instantiation, the uniformly random input is mapped to a non-uniform vector in the ring.

it samples an $x_i \xleftarrow{R} \mathcal{X}$, maps it via $\mathbf{x}_i \leftarrow \mathsf{map}(x_i)$, and computes the corresponding authorized shift, denoted by α_i (recall that there always exists an efficiently computable shift, by the correctness property of ShCPRF). Then for all $b \in \{0,1\}$ and all $\alpha \in S_b \setminus \{\alpha_i\}$, it calls its evaluation oracle with $\mathbf{x}_i$ and shift α, and receives response $y_{i,\alpha}$. For the remaining authorized entry pair $(\mathbf{x}_i, \alpha_i)$ it computes $y_{i,\alpha_i} \leftarrow \mathsf{CEval}(\mathsf{csk}, \mathbf{x}_i)$, as in the previous hybrid. It then assembles these entries into the two lists L_0^i, L_1^i, according to whether the respective shift belongs to S_0 or S_1, and sends K_R and (x_i, L_0^i, L_1^i) to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs what $\mathcal{A}$ outputs.

Observe that $\mathcal{B}$ is an efficient algorithm, making $N \cdot (|\mathcal{R}| - 1)$ oracle queries. (Note that for each query issued by the adversary, $\mathcal{B}$ needs to perform $|\mathcal{R}| - 1$ queries to the shiftable CPRF oracle.) Moreover, if $\mathcal{B}$ is given inputs from the real game $\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{B},0}(\lambda)$, then this perfectly simulates hybrid $\mathcal{H}_1$ for $\mathcal{A}$, and if it is given inputs from the ideal game $\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{B},1}(\lambda)$ then this perfectly simulates hybrid $\mathcal{H}_2$ for $\mathcal{A}$. As such, $\mathcal{B}$ has the same advantage as $\mathcal{A}$. □

Notice that $\mathcal{H}_2$ is distributed identically to $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,1}(\lambda)$ experiment defined in Figure 3, which concludes the proof of sender security.

Receiver Security. Receiver security follows from the fact that $K_S = \mathsf{msk}$ and x_i are independent of the IPM-wPRF key $\mathbf{z}$, the x_i are sampled uniformly at random, and because f is a wPRF with range $\{0,1\}$. Formally, we prove receiver security via a sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,0}(\lambda)$ experiment defined in Figure 4, where $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{EvalS}, \mathsf{EvalR})$ are as defined in Figure 5.

Hybrid $\mathcal{H}_1$. In this hybrid, we sample a uniformly random function R from the set of all functions from $\mathcal{X}_\lambda$ to $\{0,1\}$, and generate $b_i \leftarrow R(x_i)$.

Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

Proof. The only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that $b_i = f_{\mathbf{z}}(\mathsf{map}(x_i))$ in $\mathcal{H}_0$, and $b_i = R(x_i)$ in $\mathcal{H}_1$. As the x_i's are uniformly random (and $K_S = \mathsf{msk}$ is generated independently of $\mathbf{z}$), any distinguisher between $\mathcal{H}_0$ and $\mathcal{H}_1$ immediately yields a distinguisher against the pseudorandomness of $f_{\mathbf{z}}$. □

Hybrid $\mathcal{H}_1$. In this hybrid, each bit b_i is sampled uniformly at random: $b_i \xleftarrow{R} \{0,1\}$. Note that this hybrid is exactly $\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,1}(\lambda)$.

Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

Proof. Since R is a truly random function, $\mathcal{H}_2$ and $\mathcal{H}_1$ are perfectly indistinguishable conditioned on all x_i's being distinct. By a straightforward union bound, since all x_i's are sampled randomly from $\mathcal{X}$, the condition is satisfied except with probability at most $N^2/|\mathcal{X}_\lambda|$, which is negligible in λ, because $|\mathcal{X}_\lambda|$ is exponential in λ (we require this, because it is necessary for the wPRF security anyway). □

This concludes the proof of receiver security and the proof of Theorem 2.

■

D.3 Proof of Theorem 3

Proof. We prove each property in turn.

Correctness. Correctness follows directly from the proof of Theorem 1 (correctness proof of the non-updatable ShCPRF, cf. Appendix D.1).

Updatable correctness. Consider an arbitrary $\mathsf{csk} := (\mathbf{k'_0}, \mathbf{Z_1}) \in \mathcal{R}^m \times \mathcal{R}^{m \times n}$, a constraint $\mathbf{z} \in \mathcal{R}^n$, and an input $\mathbf{x} \in \mathcal{R}^n$. Let $\alpha := \langle \mathbf{z}, \mathbf{x} \rangle$. Given $\mathsf{ltsk} := \Delta \xleftarrow{R} \mathcal{R}^m$, we have $\mathsf{msk}' = (\mathsf{ltsk}, \mathsf{esk}')$ with $\mathsf{esk}' = (\mathbf{Z'_0} := \mathbf{Z_1} + \Delta \mathbf{z}^\top, \mathbf{k'_0})$. Then, denoting $\mathbf{k}_\alpha := \mathbf{k'_0} + \mathbf{Z'_0}\mathbf{x} - \Delta\alpha$, it holds that $\mathbf{k}_\alpha = \mathbf{k'_0} + (\mathbf{Z_1} + \Delta\mathbf{z}^\top)\mathbf{x} - \Delta\alpha = \mathbf{k'_0} + \mathbf{Z_1}\mathbf{x} + \cancel{\Delta(\langle \mathbf{z}, \mathbf{x} \rangle - \alpha)} = \mathbf{k'_0} + \mathbf{Z_1}\mathbf{x}$. Therefore, $F_{\mathbf{k}_\alpha}(\mathbf{x}) = \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{z}, \mathbf{x})$.

(1-key, selective, ℓ-instance) updatable security. We prove security by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

Hybrid $\mathcal{H}_0$. This hybrid consists of the (1-key, selective, ℓ-instance) updatable security game. Specifically, at the start of the game, the challenger is given the constraints $\mathbf{z}_i$ and the constrained keys $\mathsf{csk}_i := (\mathbf{k_0}_i, \mathbf{Z_1}_i)$, where $\mathbf{k_0}_i \in \mathcal{R}^m$ and $\mathbf{Z_1}_i \in \mathcal{R}^{m\times n}$ for all $i \in [\ell]$. Then, the challenger computes the updated master secret key as $\mathsf{msk}'_i := (\mathsf{ltsk} := \Delta, \mathsf{esk}_i := (\mathbf{k_0}_i, \mathbf{Z_0}_i))$, where $\mathbf{Z_0}_i := \mathbf{Z_1}_i + \Delta\mathbf{z}_i^\top$ with $\Delta \xleftarrow{R} \mathcal{R}^m$.

Hybrid $\mathcal{H}_1$. In this hybrid, we modify the updatable security game $\mathcal{H}_0$ as follows: given $(\mathbf{z}_i, \mathsf{csk}_i)_{i\le\ell}$, the challenger does not sample Δ anymore and instead interacts with the oracle $\mathcal{O}_{\mathsf{rka}}$ from the proof of Theorem 1. Similarly to the proof of Theorem 1, for evaluation query $(\mathbf{x}, \alpha, i)$ from $\mathcal{A}$ with $C_{\mathbf{z}_i}(\mathbf{x}, \alpha) \neq 0$, the challenger computes $a := \langle \mathbf{z}_i, \mathbf{x}\rangle - \alpha$ and $\mathbf{b} := \mathbf{k_0}_i + \mathbf{Z_1}_i\mathbf{x}$.

- If $b = 0$, the challenger queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi, \mathbf{x})$, where $\phi\colon \mathbf{u} \mapsto a\mathbf{u} + \mathbf{b}$ and forwards the response to $\mathcal{A}$.
- If $b = 1$, the challenger returns $y \leftarrow R(i, \mathbf{x}, \alpha)$.

Observe that for each query $(\mathbf{x}, \alpha, i)$, it holds that $\mathbf{k}_\alpha = \mathbf{k_0}_i + \mathbf{Z_1}_i\mathbf{x} + \Delta(\langle \mathbf{z}_i, \mathbf{x}\rangle - \alpha)$ is equal to $\phi(\mathbf{x}) = a\mathbf{x} + \mathbf{b}$. Hence, the answers of the challenger to all queries issued by $\mathcal{A}$ in $\mathcal{H}_1$ are computed identically to $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This hybrid consists of the RKA security game for $\mathcal{F}$ with respect to affine related key derivation functions Φ_{aff}.

Claim. If there exits an efficient adversary $\mathcal{A}$ for $\mathcal{H}_1$ that wins with non-negligible advantage, then there exists an efficient Φ_{aff}-restricted adversary $\mathcal{B}$ that wins the $\mathcal{H}_2$ game (RKA security game) with the same advantage as $\mathcal{A}$.

Proof. The challenger in $\mathcal{H}_1$ is already playing the role of a Φ_{aff}-restricted adversary when querying the oracle $\mathcal{O}_{\mathsf{rka}}$ to answer the evaluation queries. The reduction to RKA security of $\mathcal{F}$ is therefore immediate. □

Pseudorandomness. The proof of pseudorandomness follows directly from the proof of Theorem 1.

This concludes the proof of Theorem 3. ■

D.4 Proof of Theorem 5

Generation phase. For every corrupted sender S_k, the simulator reconstructs $\mathsf{ltsk}_k \leftarrow \Delta \in \mathcal{R}^m$ (using their random tape) and sends (ltsk_k, k) on their behalf to $\mathcal{F}_{\mathsf{PKS}}$. For every corrupted receiver R_l, the simulator reconstructs the constraint $C_l \leftarrow \mathbf{z}$ (using their random tape) and sends (C_l, l) to $\mathcal{F}_{\mathsf{PKS}}$ on their behalf. We now emulate each key derivation phase between a sender $S := S_k$ and a receiver $R := R_l$.

Case 1: Both parties are honest. For each $i \in [m]$, the sender computes $z_0^i \leftarrow \left\lceil \left\langle \mathsf{pk}_R, (\Delta_i, s_0^i)\right\rangle \right\rfloor_t \in \mathcal{P}$, the receiver computes $z_1^i \leftarrow \left\lceil \mathsf{pk}_S^i \cdot s_1 \right\rfloor_t \in \mathcal{P}$ and they each parse the first n coefficients of their respective polynomial as vectors $\mathbf{z}_0^i, \mathbf{z}_1^i \in \mathbb{Z}_t^n$. Observe that $\left\langle \mathsf{pk}_R, (\Delta_i, s_0^i)\right\rangle$ and $\mathsf{pk}_S^i \cdot s_1$ are in fact noisy additive shares of $\Delta_i \cdot z$ over $\mathcal{P}$, since we have that

$$\begin{aligned}
&\left\langle \mathsf{pk}_R, \left(\Delta_i, s_0^i\right)\right\rangle - (\mathsf{pk}_S^i \cdot s_1)\\
&= \Delta_i \cdot z + \Delta_i \cdot a_0 s_1 + \Delta_i \cdot e_1 + s_0^i a_1 s_1 + s_0^i e_1' - \Delta_i \cdot a_0 s_1 - s_0^i a_1 s_1 - e_0^i s_1\\
&= \Delta_i \cdot z + \underbrace{\Delta_i \cdot e_1 + s_0^i e_1' - e_0^i s_1}_{\text{noise}} \approx \Delta_i \cdot z.
\end{aligned}$$

Recall that the coefficients of z are $(q/t)\cdot(\mathbf{z}\|0^{\eta-n})$, while $\Delta_i \in \mathbb{Z}_t$ (where $t \ll q$), and $e_1, s_0^i, e_1', e_0^i, s_1$ are all drawn from χ, which we take to be a discrete Gaussian with standard deviation σ. By basic concentration inequalities, we have that $B = 3\cdot(8\sigma)^2\cdot\eta$ is a bound on the magnitude of $\Delta\cdot e_1 + s_0^i e_1' - e_0^i s_1$ with overwhelming probability. Further note that by the normal form of RingLWE, the public keys pk_R and pk_S look random, so the shares $\left\langle \mathsf{pk}_R, (\Delta_i, s_0^i)\right\rangle$ and $\mathsf{pk}_S^i \cdot s_1$ do as well. Hence by the rounding lemma (Lemma 1), since we set $q = t\cdot B\cdot n\cdot m\cdot 2^{40}$, the probability that a single component is rounded incorrectly is at most $\frac{1}{nm}\cdot 2^{-40}$. Thus, by the union bound over all $n\cdot m$ components of the shares,

for all $i \in [m]$ we have that (after parsing the ring elements) $\mathbf{z}_0^i - \mathbf{z}_1^i = \Delta_i \cdot \mathbf{z} \in \mathbb{Z}_t^n$, with very high probability.

Case 2: Sender S is corrupted. The view of the corrupted sender is $(a_0, a_1, z + s_1 a_0 + e_1, s_1 a_1 + e_1')$ where $a_0, a_1 \xleftarrow{R} \mathcal{P}$ and $s_1, e_1, e_1' \xleftarrow{R} \chi$. By the normal form of RingLWE we have that $(a_0, a_1, s_1 a_0 + e_1, s_1 a_1 + e_1') \approx_c (a_0, a_1, u_0, u_1)$ where $u_0, u_1 \xleftarrow{R} \mathcal{P}$. Hence, the simulator emulates the view of the sender using a random $\mathsf{pk}_R := (u_0, u_1) \xleftarrow{R} \mathcal{P}^2$. Eventually, the simulator recovers $\mathsf{sk}_S = (\Delta, s_0^1, \ldots, s_0^m)$, computes $\mathsf{esk} := (\mathbf{k}_0, \mathbf{Z}_0) \leftarrow \mathsf{KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_R)$ and sends esk on behalf of the ideal adversary to the functionality. The output of R in the ideal world is equal to $(\mathbf{k}_0, \mathbf{Z}_0 - \Delta \mathbf{z}^\top)$, which is identical to R's output in the real world (by the same analysis as for Case 1).

Case 3: Receiver R is corrupted. In each key derivation phase with a sender, the view of the corrupted receiver is $(a_0, a_1, \mathbf{k}_0, (\Delta_i \cdot a_0 + s_0^i + e_0^i)_{i \in [m]})$ where $a_0, a_1 \xleftarrow{R} \mathcal{P}, \mathbf{k}_0 \xleftarrow{R} \mathcal{R}^m$, and for each $i \in [m]$, $s_0^i, e_0^i \xleftarrow{R} \chi$. By the normal form RingLWE assumption, this is computationally indistinguishable from $(a_0, a_1, \mathbf{k}_0, (u^i)_{i \in [m]})$ where $a_0, a_1 \xleftarrow{R} \mathcal{P}$ and for each $i \in [m]$, $u_i \xleftarrow{R} \chi$, via a straightforward hybrid argument. Hence, the simulator emulates the view of the receiver using a random $\mathsf{pk}_S := (\mathbf{k}_0, (u^i)_{i \in [m]}) \xleftarrow{R} \mathcal{R}^m \times \mathcal{P}^m$. Eventually, the simulator recovers $\mathsf{sk}_R = (\mathbf{z}, s_1)$, computes $\mathsf{csk} := (\mathbf{k}_0, \mathbf{Z}_1) \leftarrow \mathsf{KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_S)$, and sends csk on behalf of the ideal adversary to the functionality. The output of S in the ideal world is equal to $(\mathbf{k}_0, \mathbf{Z}_1 + \Delta \mathbf{z}^\top)$, which is identical to S's output in the real world (by the same analysis as for Case 1).