Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation

Anonymous Author(s)

ABSTRACT

We consider the problem of securely generating useful instances of two-party correlations, such as many independent copies of a random oblivious transfer (OT) correlation, using a small amount of communication. This problem is motivated by the goal of secure computation with *silent preprocessing*, where a low-communication input-independent setup, followed by local ("silent") computation, enables a lightweight "non-cryptographic" online phase once the inputs are known.

Recent works of Boyle et al. (CCS 2018, Crypto 2019) achieve this goal with good concrete efficiency for useful kinds of two-party correlations, including OT correlations, under different variants of the Learning Parity with Noise (LPN) assumption, and using a small number of "base" oblivious transfers. The protocols of Boyle et al. have several limitations. First, they require a large number of communication rounds. Second, they are only secure against *semi-honest* parties. Finally, their concrete efficiency estimates are not backed by an actual implementation. In this work we address these limitations, making three main contributions:

- **Eliminating interaction.** Under the same assumption, we obtain the first concretely efficient *2-round* protocols for generating useful correlations, including OT correlations, in the semi-honest security model. This implies the first efficient 2-round OT extension protocol of any kind and, more generally, protocols for *non-interactive secure computation* (NISC) that are concretely efficient and have the silent preprocessing feature.
- **Malicious security.** We provide security against malicious parties without additional interaction and with only a modest overhead; prior to our work, no similar protocols were known with any number of rounds.
- **Implementation.** Finally, we implemented, optimized, and benchmarked our 2-round OT extension protocol, demonstrating that it offers a more attractive alternative to the OT extension protocol of Ishai et al. (Crypto 2003) in many realistic settings.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**;

KEYWORDS

secure computation; oblivious transfer extension; pseudorandom correlation generators; active security

1 INTRODUCTION

There is a large body of work on optimizing the concrete efficiency of secure computation protocols via input-independent preprocessing. By securely generating many instances of simple correlations, one can dramatically reduce the online communication and computation costs of most existing protocols.

To give just one example, multiple independent instances of a random oblivious transfer[1] (OT) correlation can be used for secure two-party computation of Boolean circuits in the semi-honest model, with communication cost of only two bits per party per (nonlinear) gate, and with computation cost that is comparable to computing the circuit with no security requirements at all [9, 32, 43]. Thus, assuming a fast communication network, protocols based on correlated randomness can achieve near-optimal performance.

The main challenge in applying this approach is the high concrete cost of securely generating the correlated randomness. Traditional solutions involve carefully optimized special-purpose secure computation protocols that have a high communication cost for each instance of the desired correlation [12, 22]. This holds even for the case of OT correlations, for which relatively fast OT extension techniques are known [8, 37, 41]. Moreover, even if offline communication is cheap, the cost of *storing* large amounts of correlated randomness for each party with whom a future interaction *might* take place can be significant.

Motivated by the limitations of traditional approaches for generating and storing correlated randomness, the notion of a *pseudorandom correlation generator* (PCG) was recently proposed and studied by Boyle et al. [16, 17]. The goal of a PCG is to compress long sources of correlated randomness without violating security. More concretely, a (two-party) PCG replaces a target two-party correlation, say many independent OT correlation instances, by a pair of *short* correlated keys, which can be "silently" expanded *without any interaction*. The process of generating the correlated keys and locally expanding them should emulate an ideal process for generating the target correlation not only from the point of view of outsiders, but also from the point of view of *insiders* who can observe one of the two keys. Among other results, the aforementioned works of Boyle et al. [16, 17] obtain concretely efficient constructions of PCGs for OT correlations and vector-OLE (VOLE) correlations [4, 39, 52] based on variants of the Learning Parity with Noise assumption [13]. These PCG constructions are motivated by the goal of secure computation with *silent preprocessing*, where a low-communication input-independent setup, followed by local ("silent") computation, enables a lightweight "non-cryptographic" online phase once the inputs are known.

However, towards realizing this goal, one major challenge remains: how can the pair of keys be securely generated? While the keys are short, their sampling algorithm is quite complex and involves multiple invocations of cryptographic primitives. Thus, even applying the fastest general-purpose protocols (e.g., [40]) for generating these keys incurs a very significant overhead.

An alternative approach for distributing the PCG key generation, suggested in [16, 17], relies on a recent special-purpose protocol of

[1] In (a single instance of) a random OT correlation, one party obtains a pair of random bits (more generally, strings) (s_0, s_1) and the other obtains the pair (r, s_r) for a random bit r.

Doerner and shelat [24]. This protocol only makes a black-box use of symmetric cryptography and a small number of oblivious transfers, and hence it is also concretely efficient. Using this protocol for distributing the key generation of a PCG for OT correlations, Boyle et al. [17] obtained a "silent OT extension" protocol that generates (without any trusted setup) a large number of pseudo-random OTs from a small number of "base" OTs using a low-communication setup followed by silent key expansion [17]. This can be generalized to other useful correlations.

While the silent OT extension protocol from [17] and other protocols obtained using this approach have good concrete efficiency, they also have several limitations. First, they require a large number of communication rounds that grows (at least) logarithmically with the output length. Second, they are only secure against *semi-honest* parties. Finally, their concrete efficiency estimates are not backed by an actual implementation, and ignore possible cache-misses and other system- and network-related sources of slowdown.

1.1 Our Contribution

In this work, we address the above limitations by making the following contributions.

Two-Round Silent OT Extension. We present the first *concretely efficient* two-round OT extension protocol, based on a variant of the LPN assumption. The protocol has a silent preprocessing feature, allowing the parties to push the bulk of the computational work to an offline phase. It can be used in two modes: either a random-input mode, where the communication complexity is sublinear in the outputs length, or a chosen-input mode, where the communication is comparable to the total input length. This is the case even for the more challenging case of 1-bit OT, for which standard OT extension techniques that make a black-box use of symmetric cryptography [8, 37, 41, 44] have a high communication overhead compared to the input length.

Our OT extension protocol bypasses a recent impossibility result of Garg et al. [28] on 2-round OT extension due to the use of the LPN assumption. While our construction (inevitably) does not fall into the standard black-box framework considered [28], it still has a black-box flavor in that it only invokes a syndrome computation of *any* error-correcting code for which the LPN assumption holds. We remark that beyond its concrete efficiency, our OT extension protocol can be used with a conservative variant of (binary) LPN that has a relatively high noise rate and is not known to imply public-key encryption, let alone oblivious transfer [3]. We also extend the 2-round OT result to other useful instances of *non-interactive secure computation* (NISC) [38] with silent preprocessing.

Malicious Security. We present simple, practical techniques for secure distributed setup of puncturable PRF keys with a weak form of malicious security. This suffices to upgrade our semi-honest protocols to malicious security, at a very low cost. Our main protocols in this setting have 4 rounds of interaction, but this can be reduced to 2 rounds using the Fiat-Shamir transform. We can also use this to obtain maliciously secure silent NISC or two-round OT extension on chosen inputs. These protocols are based on slightly stronger variants of LPN, where the adversary is allowed a single query to a one-bit leakage function on the error vector.

Implementation. We demonstrate the efficiency of our constructions with an implementation of our random OT extension protocol. The most costly part of the implementation is a large matrix-vector multiplication, which comes from applying the LPN assumption. We optimize this using quasi-cyclic codes, similarly to several recent, candidate post-quantum secure cryptosystems, and present different tradeoffs with parameter choices. Our protocols have a very low communication overhead and perform significantly faster than previous, state-of-the-art protocols [8, 37, 41] in environments with restricted bandwidth. For instance, in a 100Mbps WAN setting, we are around 5x faster, and this improves to 47x in a 10MBps WAN. This is because, while our computational costs are around an order of magnitude higher, we need around 1000–2000 times less communication than the other protocols.

Applications. As well as the new application to NISC with silent preprocessing, our protocols can be applied to a range of secure computation tasks. Below we mention just a few areas where we expect our OT extension to have an impact.

- **Semi-honest MPC.** In the semi-honest "GMW protocol" [32], the correlated randomness needed to evaluate a Boolean circuit can be obtained from two random OTs per AND gate. Plugging in our random OT extension, we obtain a practical 2-PC protocol where each party communicates just 2 bits per AND gate on average. This is around 30x less communication than the state-of-the-art [23].
- **Malicious secure MPC.** Protocols based on authenticated garbling [59, 60] are currently the state-of-the-art in maliciously secure MPC for binary circuits, particularly in a high-latency scenario. The main cost in these protocols comes from a preprocessing phase, where the parties use a large number of random, correlated oblivious transfers to produce correlated randomness. Our protocol can produce the same kind of oblivious transfers with almost zero communication, and we estimate this could reduce the *overall* communication in authenticated garbling protocols by around an order of magnitude.
- **Private set intersection (PSI).** In circuit-based PSI, a generic 2-PC protocol is used to first compute a secret-sharing of the intersection of two sets, and then perform some useful computation on the result [36, 54, 55]. With the improvements to GMW mentioned above, we can expect to obtain a similar reduction in communication for these families of PSI protocols.

1.2 Technical Overview

We now give an overview of our silent constructions in the semi-honest and malicious settings. For simplicity, we focus here on the case of 1-out-of-2 oblivious transfer.

We start by recalling the high-level idea of the pseudorandom correlation generators for vector-OLE and OT from [16, 17]. These constructions distribute a pair of seeds to a sender and a receiver, who can then locally expand the seeds to produce many instances of pseudorandom OT or vector-OLE. To do so, they use two main ingredients: a variant of the LPN assumption, and a method for the sender and receiver to obtain a *compressed* form of random secret shares $\vec{v}_0, \vec{v}_1$, respectively, satisfying

$$\vec{v}_1 = \vec{v}_0 + \vec{e} \cdot x \in \mathbb{F}_{2^\lambda}^N \quad (1)$$

where $\vec{e} \in \{0,1\}^N$ is a random, sparse vector held by the receiver, and $x \in \mathbb{F}_{2^\lambda}$ is a random field element held by the sender.

Given this, the shares can be randomized by taking a public, binary matrix H that compresses from N down to $n < N$ elements, and locally multiplying each share with H. This works because $\vec{u} = \vec{e} \cdot H$ is pseudorandom under a suitable variant of LPN. Writing $\vec{v} = \vec{v}_0 \cdot H$ and $\vec{w} = \vec{v}_1 \cdot H$, from (1) we then get $\vec{w} = \vec{v} + \vec{u}x$. This can be seen as a set of random *correlated OTs*, where $u_i \in \{0,1\}$ are the receiver's choice bits, and $(v_i, v_i + x)$ are the sender's strings, of which the receiver learns w_i. These can be locally converted into random string-OTs with a standard hashing technique [37].

To obtain a compressed form of the shares in (1), the constructions of [16, 17] used a *distributed point function* (DPF). Our first observation is that the distributed point function is not needed, and can be replaced with a *puncturable pseudorandom function.* A puncturable pseudorandom function (PPRF) is a PRF F such that given an input x, and a PRF key k, one can generate a punctured key $\mathsf{k}\{x\}$ which allows evaluating F at every point except for x, and does not conceal any information about the value $F(\mathsf{k}, x)$. A PPRF can be built from any length-doubling pseudorandom generator, using a binary tree-based construction [15, 18, 42].

In the setup procedure, we will give the sender a random key k and x, and give to the receiver a random point $\alpha \in \{1, \ldots, N\}$, a punctured key $\mathsf{k}\{\alpha\}$, and the value $z = F(\mathsf{k}, \alpha) + x$. Given these seeds, the sender and receiver can now define the expanded outputs, for $i = 1, \ldots, n$:

$$\vec{v}_0[i] = F(\mathsf{k}, i), \quad \vec{v}_1[i] = \begin{cases} F(\mathsf{k}, i) & i \neq \alpha \\ z & \text{otherwise} \end{cases}$$

These immediately satisfy (1), with $\vec{e}$ as the α-th unit vector. To obtain sharings of sparse $\vec{e}$ with, say, t non-zero coordinates, as needed to use LPN, we repeat this t times and XOR together all t sets of outputs.

Conceptually, this construction is simpler than using a DPF, and moreover, as we now show, it brings several efficiency advantages.

Two-Round Setup of Puncturable PRF Keys. We present a simple, two-round protocol for distributed the above setup with semi-honest security, inspired by the DPF setup protocol of Doerner and shelat [24]. The core of our protocol is the following procedure. For each of t secret LPN noise coordinates $\alpha_j \in [N]$ known to the receiver, the sender generates a fresh PRF key k_j, and wishes to obliviously communicate a punctured key $\mathsf{k}_j\{\alpha_j\}$ and hardcoded punctured output $z_j = PRF(\mathsf{k}_j, \alpha) + x$ to the receiver. Combined, this yields a secret sharing of the vector $x \cdot \vec{e}$, as required. To do so, for each $\mathsf{k}\{\alpha\}$, the parties made use of $\ell = \log N$ parallel OT executions: the sender's ℓ message pairs correspond to appropriate sums of partial evaluations from a consistent GGM PRF tree and his secret value x, and the receiver's ℓ selection bits correspond to the bits of his chosen path α.

Compared with previous works based on distributed point functions [16, 17, 24], the number of rounds of interaction collapses from $O(\log N)$ to just two, given any two-round OT protocol.

Two-Round OT Extension and Silent NISC. We observe that in the two-round setup, the receiver can *already compute* part of its output before sending the first round message. In the case of OT, this part corresponds to its random vector of choice bits $\vec{u}$. This means that the receiver can already *derandomize* its OT outputs in the first round, by sending in parallel with its setup message the value $\vec{u} + \vec{c}$, where $\vec{c}$ is its *chosen* input vector. Since the sender can compute its random OT outputs after the first round, this leads to a two-round OT extension protocol that additionally enjoys the "silent preprocessing" feature of pushing the bulk of the computation to an offline phase, before the inputs are known. This can be generalized from OT to VOLE and other useful instances of *non-interactive secure computation* (NISC) [38], simultaneously inheriting the silent preprocessing feature from the PCG and the interaction feature from an underlying NISC protocol. See Section 3 for a more detailed discussion of our new notion of NISC with silent preprocessing.

Maliciously Secure Setup. In the above semi-honest setup procedure, a malicious *receiver* has no cheating space; altered selection bits merely correspond to a different choice of noise coordinate $\alpha' \in [N]$. However, a malicious *sender* may generate message pairs inconsistent with any correct PRF evaluation tree, or use inconsistent inputs x across the t executions (in which case the outputs are not valid shares of $x \cdot \vec{u}$ for any single x). For example, by injecting errors into one of the two messages within an OT message pair, the sender can effectively "guess" and learn a bit of α, and will go unnoticed if his guess is correct.

We demonstrate that with small overhead, we can restrict a malicious sender to *only* such selective-failure attacks. This is formalized via an ideal functionality where the adversarial sender can send a guess range $I \subseteq [N]$ for α, a "getting caught" predicate is tested as a function of the receiver's true input, and the functionality either aborts or delivers the output accordingly. We then show that paired with an interactive leakage notion for LPN, this suffices to give us PCG setup protocols for VOLE and OT with malicious security.

Our basic maliciously secure protocols have 4 rounds, but this can be compressed to two rounds with the Fiat-Shamir transform, in the random oracle model. Just as in the semi-honest protocols, we can convert the setup protocols into NISC protocols, this time under a slightly stronger variant of LPN with one bit of *adaptive* leakage on the error vector, obtaining two-round OT extension with malicious security.

2 PRELIMINARIES

2.1 Puncturable Pseudorandom Function

Pseudorandom functions (PRF) are keyed functions which are indistinguishable from truly random functions, have been introduced in [31]. A *puncturable pseudorandom function* (PPRF) is a PRF F such that given an input x, and a PRF key k, one can generate a *punctured* key, denoted $k\{x\}$, which allows evaluating F at every point except for x, and does not conceal any information about the value $F.\mathsf{Eval}(k, x)$. PPRFs have been introduced in [15, 18, 42].

Definition 2.1 (t-Puncturable Pseudorandom Function). A puncturable pseudorandom function (PPRF) with key space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$, is a pseudorandom function F with an additional

Experiment Exp-s-pPRF

Setup Phase. The adversary $\mathcal{A}$ sends a size-t subset $S^* \in \mathcal{X}$ to the challenger. When it receives S^*, the challenger picks $K \xleftarrow{\$} F.\mathsf{KeyGen}(1^\lambda)$ and a random bit $b \xleftarrow{\$} \{0,1\}$.

Challenge Phase. The challenger sends $K\{S^*\} \leftarrow F.\mathsf{Puncture}(K, S^*)$ to $\mathcal{A}$. If $b = 0$, the challenger additionally sends $(F(K,x))_{x \in S^*}$ to $\mathcal{A}$; otherwise, if $b = 1$, the challenger picks t random values ($y_x \xleftarrow{\$} \mathcal{Y}$ for every $x \in S^*$) and sends them to $\mathcal{A}$.

Figure 1: Selective security game for puncturable pseudo-random functions. At the end of the experiment, $\mathcal{A}$ sends a guess b' and wins if $b' = b$.

punctured key space $\mathcal{K}_p$ and three probabilistic polynomial-time algorithms ($F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval}$) such that

- $F.\mathsf{KeyGen}(1^\lambda)$ outputs a random key $K \in \mathcal{K}$,
- $F.\mathsf{Puncture}(K, x)$, on input a ley $K \in \mathcal{K}$, and a subset $S \subset \mathcal{X}$ of size t, outputs a punctured key $K\{S\} \in \mathcal{K}_p$,
- $F.\mathsf{Eval}(K\{S\}, x)$, on input a key $K\{S\}$ punctured at all points in S, and a point x, outputs $F(K,x)$ if $x \notin S$, and $\perp$ otherwise,

such that no probabilistic polynomial-time adversary wins the experiment Exp-s-pPRF represented on Figure 1 with non-negligible advantage over the random guess.

A PPRF can be constructed from any length-doubling pseudo-random generator, using the GGM tree-based construction [15, 18, 31, 42]. The construction proceeds as follows: On input a key K and a point x, set $K^{(0)} \leftarrow K$ and perform the following iterative evaluation procedure: for $i = 1$ to $\ell \leftarrow \log|x|$, compute $(K_0^{(i)}, K_1^{(i)}) \leftarrow G(K^{(i-1)})$, and set $K^{(i)} \leftarrow K_{x_i}^{(i)}$. Output $K^{(\ell)}$. This procedure creates a complete binary tree with edges labeled by keys; the output of the PRF on an input x is the key labeling the leaf at the end of the path defined by x from the root of the tree.

- $F.\mathsf{KeyGen}(1^\lambda)$: output a random seed for G.
- $F.\mathsf{Puncture}(K, z)$: on input a key $K \in \{0,1\}^k$ and a point x, apply the above procedure and return $K\{x\} = (K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$.
- $F.\mathsf{Eval}(K\{x\}, x')$, on input a punctured key $K\{x\}$ and a point x, if $x = x'$, output $\perp$. Otherwise, parse $K\{x\}$ as $(K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$ and start the iterative evaluation procedure from the first $K_{1-x_i}^{(i)}$ such that $x'_i = 1 - x_i$.

To obtain a t-puncturable PRF with input domain $[n]$, one can simply run t instances of the above puncturable PRF and set the output of the PRF to be the bitwise xor of the output of each instance. With this construction, the length of a key punctured at t points is $t\lambda \log n$, where λ is the seed size of the PRG.

2.2 Learning Parity with Noise

In this work, we rely on variants of the Learning Parity with Noise (LPN) assumption [13] over either $\mathbb{F}_2$ or a large finite field $\mathbb{F}$, where the noise is assumed to have a small Hamming weight. Similar assumptions have been used in the context of secure arithmetic computation [4, 25, 29, 39, 52]; unlike most of these works, the flavors of LPN on which we rely do not require the underlying code to have an algebraic structure and are thus not susceptible to algebraic (list-) decoding attacks. More detailed preliminaries on LPN are given in Appendix B. We introduce below the dual-LPN assumption with code matrix H and error distribution $\mathcal{D}(\mathcal{R})$.

Definition 2.2 (dual LPN). Let $\mathcal{D}(1^\lambda, \mathcal{R})$ be a PPT algorithm sampling a noise distribution over $\mathcal{R}^N$, where $\mathcal{R}$ is a finite ring. We will consider either *uniform noise*, where $\mathcal{D}$ outputs a uniformly random vector of length $N(\lambda)$ and Hamming weight $t(\lambda)$, or *regular noise*, where the output is partitioned into $t(\lambda)$ blocks, each containing a random weight-1 vector. For $n = n(\lambda), N = N(\lambda)$ (where $N > n$), $H \in \mathcal{R}^{N \times n}$ (generated by a PPT code generation algorithm $\mathcal{H}(1^\lambda, \mathcal{R})$) and $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathcal{H}, \mathcal{R})$-dual-LPN$(N, n)$ assumption states that

$$\{(H, \vec{b}) \mid \vec{e} \xleftarrow{\$} \mathcal{D}(\mathcal{R}), \vec{b} \leftarrow \vec{e} \cdot H\} \overset{c}{\approx} \{(H, \vec{b}) \mid \vec{b} \xleftarrow{\$} \mathcal{R}^n\}.$$

We will slightly abuse our notations and directly refer to the $(\mathcal{D}, H, \mathcal{R})$-dual-LPN$(N, n)$ for a fixed matrix H sampled from $\mathcal{H}$ when it is clear from the context.

Attacks on LPN. The LPN settings which we will employ in this work is identical to the settings used in [16, 17]. We refer the reader to these works for a summary of the existing attacks on LPN, thorough estimates of their asymptotic complexity on the specific LPN instances we consider, and concrete estimates of the exact parameters needed to achieve an appropriate security level. We will rely on their analysis (see also Appendix B) when analyzing the concrete efficiency of our constructions in Section 7. We will refine it for our purpose, taking into account the DOOM attack [57] targetting LPN variants with quasi-cyclic matrices (which we use for improving computational efficiency) and analyzing the resistance against the BJMM attack [11], which is the state of the art variant of Prange's information set decoding attack [56].

2.3 Secure Computation and NISC

We use standard definitions of (composable) secure two-party computation. Our protocols can be analyzed and used either in a standalone setting, as formalized in [19, 30], or in a UC setting [20, 38, 53]. It will be convenient to cast our protocols in a hybrid model that allows parallel calls to an ideal oblivious transfer functionality. These calls can be instantiated by any composable OT protocol (e.g., the "PVW protocol" [53] when considering UC security against malicious adversaries in the CRS model). We use λ to denote a computational security parameter, which we view as a public parameter that is available to all algorithms even when not explicitly stated.

We will specifically be interested in 2-round protocols for "sender-receiver functionalities" that take an input x from a receiver R and input y from a sender S, and deliver an output $f(x, y)$ to R. The communication consists of a single message from the receiver to the sender followed by a single message from the receiver to the sender. Such protocols can be viewed as being *non-interactive* in that the receiver can publish its message $\hat{x}$ (which depends only on its input x) and then go offline, before even knowing who the sender will be. Then $\hat{x}$ can be used by any sender S (in fact, in some cases even multiple senders) by sending the encrypted output $\hat{z}$

to the receiver's mailbox. We use the term *non-interactive secure computation* from [38] (NISC for short) to highlight this qualitative advantage. When described in the OT-hybrid model, NISC protocols involve only one round of parallel OT calls. They can additionally involve a message from R to S and a message from S to R, as long as these messages (in an honest execution) do not depend on outputs of the OT oracle. Such NISC protocols in the OT-hybrid model can be converted into NISC protocols in the plain model (or CRS model for malicious security) using any 2-round (parallel-)OT protocol.

2.4 Pseudorandom Correlation Generators

A (two-party) pseudorandom correlation generator (PCG) securely generates long correlated pseudo-randomness from a pair of correlated keys. Defining a PCG requires care, since the natural simulation-based definition is not realizable. Instead, the following relaxed definition has been proposed in [16, 17].

The ideal output distribution of a PCG is specified by a (long) target correlation (R_0, R_1), e.g., n independent instances of an OT correlation. This target correlation is specified by PPT algorithm C, called a *correlation generator*, where $C(1^\lambda)$ outputs a pair of strings. We furthermore restrict C to be *reverse-samplable* in the following sense: there exists a PPT algorithm RSample such that for $\sigma \in \{0,1\}$, the correlation obtained via:

$$\{(R'_0, R'_1) \mid (R_0, R_1) \xleftarrow{\$} C(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $C(1^\lambda)$.

Examples for standard and useful correlations, all of which are reverse-samplable, include Oblivious Transfer (OT) correlation, where R_0 includes n independent pairs of bit-strings (s_0^i, s_1^i) and R_1 includes $(c_i, s_{c_i}^i$ for random bits c_i, and Vector-OLE (VOLE) correlation over a finite field $\mathbb{F}$, where $R_0 = (\vec{u}, \vec{v})$ for random $\vec{u}, \vec{v} \in \mathbb{F}^n$, and $R_1 = (x, \vec{u}x + \vec{v})$ for random $x \in \mathbb{F}$.

Definition 2.3 (Pseudorandom Correlation Generator (PCG) [17]). Let C be a reverse-samplable correlation generator. A *pseudorandom correlation generator (PCG) for C* is a pair of algorithms (PCG.Gen, PCG.Expand) with the following syntax:

- $\mathsf{PCG.Gen}(1^\lambda)$ is a PPT algorithm that given a security parameter λ, outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.

The algorithms (PCG.Gen, PCG.Expand) should satisfy:

- **Correctness.** The correlation obtained via:

$$\{(R_0, R_1) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\}$$

  is computationally indistinguishable from $C(1^\lambda)$.
- **Security.** For corrupted party $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:

$$\{(\mathsf{k}_\sigma, R_{\bar{\sigma}}) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{\bar{\sigma}} \leftarrow \mathsf{PCG.Expand}(\bar{\sigma}, \mathsf{k}_{\bar{\sigma}})\} \text{ and}$$

$$\{(\mathsf{k}_\sigma, R_{\bar{\sigma}}) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma), R_{\bar{\sigma}} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

  where $\bar{\sigma} = 1 - \sigma$ and RSample is the reverse sampling algorithm for C.

As shown in [17], a PCG as defined above can be used as a drop-in replacement' for ideal correlated randomness generated by C in any application that remains secure even when C is replaced by the following corruptible version $\tilde{C}$. In $\tilde{C}$ the corrupted party can choose its own randomness, and the randomness of the honest party $R_{1-\sigma}$ is obtained by applying RSample. It turns out that in most concretely efficient MPC protocols that consume correlated randomness, security still holds even with this corruptible variant. In particular, this holds for the simple protocols that implement standard (chosen-input) OT or VOLE from the corresponding correlations. However, applying PCGs, the pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$ to be generated either by a trusted dealer or by a secure protocol realizing PCG.Gen.

3 PCG PROTOCOLS AND SILENT NISC

We now define two new cryptographic primitives we introduce in this work: A *pseudorandom correlation generation protocol* (*PCG protocol* for short) and a *non-interactive secure computation protocol with silent preprocessing* (*silent NISC* for short).

3.1 PCG Protocols

The above notion of PCG gives a *deterministic* procedure for securely generating long sources of correlated randomness from short but *suitably correlated* seeds. It does not explicitly address the question of generating the seeds. In the following we formalize a natural generalization of PCGs to a *low-communication* protocol for securely generating long sources of correlated randomnness *from scratch*. By "low communication" we means that the total communication complexity is sublinear in the output length.[2]

We take the following natural definition approach: a PCG protocol for an ideal correlation C is a secure two-party protocol (in the usual sense) for the *corruptible* correlated randomness functionality $\tilde{C}$ described above.

Definition 3.1 (PCG protocol). Let C be a reverse-samplable correlation generator. Define a randomized functionality $\tilde{C}$ that takes from a corrupted party σ a string $\tilde{r}_\sigma$ as input, and outputs to the honest party $\bar{\sigma}$ a string $r_{\bar{\sigma}}$ sampled by $\mathsf{RSample}(\sigma, \tilde{r}_\sigma)$. If no party is corrupted, $\tilde{C}$ outputs to both parties a fresh pair of outputs generated by C. A (two-party) PCG protocol is a two-party protocol realizing $\tilde{C}$ in which the communication complexity grows sublinearly with the output length. In the case of security against *semi-honest adversaries*, we still allow the ideal-model corrupted party (if any) to pick its input $\tilde{r}_\sigma$ for $\tilde{C}$ arbitrarily, whereas the real-model adversary must follow the protocol.

As a simple corollary of an MPC composition theorem, a PCG protocol for C can serve as a substitute for ideal correlated randomness C in any higher-level application that remains secure even when C is replaced by $\tilde{C}$. Indeed, this is the case for standard MPC protocols that rely on OT correlations or other types of simple correlations, both for semi-honest and malicious security.

A general way of obtaining a PCG protocol is by distributing the randomized key generation functionality PCG.Gen of a PCG (as in

[2] In fact, in all protocols presented in this paper, the communication complexity only grows *polylogarithmically* with the output length, under widely believed variants of the LPN assumption.

Definition 2.3) via a secure two-party computation protocol, and then locally applying PCG.Expand. Indeed, this is the approach suggested in [17] for the purpose of applying PCGs in the context of "MPC with silent preprocessing." However, our notion of a PCG protocol is less stringent than an alternative definition that requires securely emulating PCG.Gen for some PCG, while at the same time being as good for applications. We make use of this extra degree of freedom in our PCG protocols for the malicious model.

A central contribution of this work is the construction of *two-round* PCG protocols, namely ones involving only a message from R to S followed by a message from S to R. We refer to such a protocol as a *non-interactive PCG* protocol. We use the following syntax to highlight the fact that the message of R can be published as a "public key" before the sender(s) are known.

Definition 3.2 (Non-interactive PCG protocol). A *non-interactive PCG protocol* is defined by 4 algorithms with the following syntax:

- $\mathsf{R.Gen}(1^\lambda) \to (sk_\mathsf{R}, pk_\mathsf{R})$
- $\mathsf{S.Gen}(pk_\mathsf{R}) \to (sk_\mathsf{S}, m_\mathsf{S})$
- $\mathsf{R.Expand}(sk_\mathsf{R}, m_\mathsf{S}) \to r_\mathsf{R}$
- $\mathsf{S.Expand}(sk_\mathsf{S}) \to r_\mathsf{S}$

We say that the above algorithms define a non-interactive PCG protocol for a reverse-samplable correlation C if the two-round protocol they naturally define (where each party outputs the output of Expand) is a PCG protocol for C as in Definition 3.1.

In a non-interactive PCG protocol as above, the two Gen algorithms can be viewed as defining a cheap setup that results in short, correlated keys. The two Expand algorithms are used to locally perform "silent preprocessing" that generates useful correlated randomness (e.g., many instances of an OT correlation, or few instances of a long VOLE correlation). In the most useful special case of OT correlations, we will refer to a non-interactive PCG that makes a small number of parallel OT calls as a non-interactive (or 2-round) *silent OT extension protocol.*

3.2 Silent NISC

In this section we define our new notion of *non-interactive secure computation with silent preprocessing*, or *silent NISC* for short. A silent NISC protocol for f can be viewed as a "best-of-both-worlds" combination of a non-interactive PCG protocol (see Definition 3.2) and a NISC protocol (see Section 2.3). That is, a 2-round (chosen-input) secure computation protocol that supports "silent preprocessing" followed by a light-weight (and often "non-cryptographic") online phase, without additional interaction.

Combining non-interactive PCG and NISC protocols in a generic way does not achieve the above goal, since it involves 4 rounds: two to generate the correlated randomness, and two to use it. To collapse these 4 rounds into two, we rely on the following feature of our concrete non-interactive PCG constructions. For useful NISC correlations such as OT and VOLE, the receiver's piece of the correlated randomness r_R can be split into two parts: r_R^in, which is used to mask its input, and $r_\mathsf{R}^\mathsf{out}$, used to unmask the output. The key feature is that the construction allows R to locally generate r_R^in from its public key pk_R alone, independently of the sender. This enables R to prepare to a future NISC before the sender is even known.

More concretely, let $f(x, y)$ be a sender-receiver functionality with receiver input x and sender input y. Useful examples for which we get efficient solutions include: (1) n instances of string-OT; (2) bitwise-AND of two n-bit strings; (3) inner product of two length-n vectors over $\mathbb{F}$; (4) a general function f represented by a Boolean circuit, which can be efficiently and non-interactively reduced to (1) via garbled circuits (see [1, 38, 50] for such black-box reductions for the malicious model).

A *NISC protocol with silent preprocessing* (or silent NISC) for f is defined by 8 algorithms:

- $\mathsf{R.Gen}(1^\lambda) \to (sk_\mathsf{R}, pk_\mathsf{R})$
- $\mathsf{R.Expand}^\mathsf{in}(sk_\mathsf{R}) \to r_\mathsf{R}^\mathsf{in}$
- $\mathsf{S.Gen}(pk_\mathsf{R}) \to (sk_\mathsf{S}, pk_\mathsf{S})$
- $\mathsf{R.Expand}^\mathsf{out}(sk_\mathsf{R}, pk_\mathsf{S}) \to r_\mathsf{R}^\mathsf{out}$
- $\mathsf{S.Expand}(sk_\mathsf{S}) \to r_\mathsf{S}$
- $\mathsf{R.Msg}(r_\mathsf{R}^\mathsf{in}, x) \to \hat{x}$
- $\mathsf{S.Msg}(r_\mathsf{S}, \hat{x}, y) \to \hat{z}$
- $\mathsf{R.Dec}(r_\mathsf{R}^\mathsf{out}, x, \hat{z}) \to z$

The security requirement is that the 2-round protocol obtained by executing the above algorithms in any consistent order satisfies the security requirement of a (standard) NISC protocol for f.

See Figure 14 in Appendix D for a diagram illustrating the full information and computation flow of silent NISC and the dependencies between the different algorithms. The 3 Expand algorithms define the "silent preprocessing" phase, that can be executed before the inputs are known. The last 3 algorithms define the online part of the NISC protocol, which is carried out once the inputs are known. Among the four examples given above, this part is "non-cryptographic" in the first three cases, and makes a black-box use of symmetric crypto in the last one.

We will be particularly interested in silent NISC realizing many parallel OTs using few parallel OTs, which can be viewed as a non-interactive, chosen-input variant of silent OT extension. While here one cannot make the communication complexity sublinear in the input length, our goal (which we achieve both in theorem and in practice) to make the communication very close to the total input length. This is the case even for the more challenging case of 1-bit OT, for which standard OT extension techniques that make a black-box use of cryptography [8, 37, 41, 44] have a high communication overhead compared to the input length.

4 OPTIMIZED VOLE/OT CONSTRUCTION

4.1 Simplified subfield VOLE generator

We provide a construction of a PCG for subfield-VOLE correlations on Fig. 2. Recall that in subfield-VOLE, one party receives random vectors $\vec{u} \in \mathbb{F}_p^N$ and $\vec{v} \in \mathbb{F}_{p^r}^N$, while the other party gets a random $x \in \mathbb{F}_{p^r}$, and $\vec{w} = \vec{u}x + \vec{v}$. The construction follows the informal description from Section 1.2 (where we described the special case $p = 2$, which is equivalent to correlated OT), and is essentially the same as the construction in [17], with a puncturable PRF instead of a DPF. Likewise, the security analysis is essentially identical to the analysis of [17].

Construction G_{sVOLE}

PARAMETERS: $1^\lambda, n, N, t, p, r \in \mathbb{N}$, where $N > n$. A matrix $H \in \mathbb{F}_p^{N\times n}$ and a weight-t error distribution $\mathcal{D}_{t,N}$ over $\mathbb{F}_p^N$.

CORRELATION: After expansion, outputs $(\vec{u}, \vec{v}) \in \mathbb{F}_p^n \times \mathbb{F}_{p^r}^n$ and $(x, \vec{w}) \in \mathbb{F}_{p^r} \times \mathbb{F}_{p^r}^N$, where $\vec{w} = \vec{u}x + \vec{v}$.

We view $\mathbb{F}_p$ as a subfield of $\mathbb{F}_{p^r}$, via some fixed embedding and representation of field elements.

PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{p^r}$.

Gen: On input 1^λ:

(1) Sample $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $S = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$, and $y_i = e_{\alpha_i} \in \mathbb{F}_p^*$.

(2) Sample $x \xleftarrow{\$} \mathbb{F}_{p^r}$.

(3) Sample $\mathsf{k}_{\mathsf{pprf}} \xleftarrow{\$} \mathsf{PPRF.Gen}(1^\lambda)$, and $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{PPRF.Puncture}(\mathsf{k}_{\mathsf{pprf}}, S)$.

(4) For $i = 1, \ldots, t$, let $z_i \leftarrow x \cdot y_i - \mathsf{PPRF.Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha_i)$

(5) Let $\mathsf{k}_0 \leftarrow (\mathsf{k}^*_{\mathsf{pprf}}, S, \vec{y}, \{z_i\}_{i\in[t]})$ and $\mathsf{k}_1 \leftarrow (\mathsf{k}_{\mathsf{pprf}}, x)$.

(6) Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$:

(1) If $\sigma = 0$, parse k_0 as $(\mathsf{k}^*_{\mathsf{pprf}}, S, \vec{y}, \{z_i\}_i)$ and do as follows:

(a) Define $\vec{e} \in \mathbb{F}_{p^m}^N$ using $\vec{y}, \{z_i\}_i$ as above.

(b) For $j \in [N]$, define the j-th entry of vector $\vec{v}_0$ as

$$\vec{v}_0[j] = \begin{cases} z_i & \text{if } j = \alpha_i \in S \\ -\mathsf{PPRF.Eval}(\mathsf{k}^*_{\mathsf{pprf}}, j) & \text{if } j \notin S \end{cases}$$

(c) Output $(\vec{u}, \vec{v}) \leftarrow (\vec{e} \cdot H, -\vec{v}_0 \cdot H)$.

(2) If $\sigma = 1$, parse k_1 as $(\mathsf{k}_{\mathsf{pprf}}, x)$ and do as follows:

(a) Compute $\vec{v}_1 \leftarrow \mathsf{PPRF.FullEval}(\mathsf{k}_{\mathsf{pprf}})$ in $\mathbb{F}_{p^r}^N$.

(b) Output $(x, \vec{w} \leftarrow \vec{v}_1 \cdot H)$.

Figure 2: PPRF-based PCG for subfield vector-OLE

4.2 Instantiating the puncturable PRF

We use a simple puncturable PRF based on the GGM approach [31] (as defined in Section 2). To build a PPRF supporting t punctured points, we simply create t independent GGM PRFs, each punctured once. Evaluation of the final PPRF is defined by adding the evaluations of all t GGM-based PRFs.

More Efficient Puncturing Strategy. The key size for the above t-puncturable PRF is $t \cdot \lambda \log(N)$. It is possible to reduce this size to $t \cdot \lambda \log(N/t)$ with a more optimized puncturing strategy; however, this alternative construction is not compatible with our optimized distributed generation protocols of Section 5 and Section 6. It is nonetheless useful in a setting where a trusted dealer is available to distribute the PCG seeds, or where computation is not a bottleneck compared to long-term storage; for completeness, we describe it in Appendix C.

Functionality $\mathcal{F}_{\mathsf{PPRF\text{-}GGM}}$:

PARAMETERS: $1^\lambda, \ell, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $\{0,1\}^\ell$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}_{p^r}$.

INPUTS:

- S inputs $\beta \in \mathbb{F}_{p^r}$ and a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.
- R inputs $\alpha \in \{0,1\}^\ell$.

FUNCTIONALITY:

- Compute $\mathsf{k}^*_{\mathsf{pprf}} = \mathsf{PPRF.Puncture}(\mathsf{k}_{\mathsf{pprf}}, \alpha)$.
- Send $\mathsf{k}^*_{\mathsf{pprf}}$ and $t = \beta - \mathsf{PPRF.Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha)$ to R.

Figure 3: Functionality for distributing a PPRF correlation

5 SEMI-HONEST PCG PROTOCOL AND TWO-ROUND OT EXTENSION

In this section, we show how to securely compute the Gen algorithm from Fig. 2, in just 2 rounds (assuming any 2-round OT). Using the construction of [17], this leads to a distributed protocol for generating random OT correlations as well, assuming in addition a correlation-robust hash function. Then, we observe that our protocols satisfy a specific feature, which allows them to be derandomized into chosen-input VOLEs and OTs, without increasing their round complexity; this leads to 2-round OT extension and VOLE extension protocols, with silent preprocessing. Our construction relies on the GGM puncturable PRF [31] constructed from any length-doubling pseudorandom generator G.

5.1 Distributed GGM-PPRF Correlation

We first consider a functionality where a party R holds a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$ for the GGM PPRF [31], and a point $\alpha = \alpha_1 \cdots \alpha_\ell$ where $\ell = \ell(\lambda)$ is logarithmic in λ, and a party S holds a value $\beta \in \{0,1\}^\lambda$. The functionality computes and gives $k\{\alpha\}, \beta - \mathsf{PPRF.Eval}(k, \alpha)$ to R. The functionality is represented on Figure 3.

THEOREM 5.1. *Assuming a black-box access to a PRG, there exists a 2-party protocol for $\mathcal{F}_{\mathsf{PPRF\text{-}GGM}}$, with semi-honest security in the OT-hybrid model, and the following efficiency features. The computational complexity is dominated by $O(2^\ell)$ calls to a length-doubling PRG $G : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$. The interaction consists of ℓ* parallel *calls to $\mathcal{F}_{\mathsf{OT}}$ and has communication complexity $\lambda + (3\lambda + 1)\ell$.*

We represent the protocol $\Pi_{\mathsf{PPRF\text{-}GGM}}$ satisfying the requirements of Theorem 5.1 on Figure 4. At a high-level, our protocol proceeds as follows: at each level i of the GGM tree, the holder of the PRF key k computes the XOR i^i_R of all odd-numbered nodes, and the XOR t^i_L of all even-numbered nodes. Using a single 1-out-of-2 OT, the receiver obtains one of (t^i_L, t^i_R). The protocol maintains the invariant that at the level $i-1$, the receiver can compute (from the previously stored information) all node values except one, implying that at level i, the receiver can compute all node values except two; recovering one of (t^i_L, t^i_R) allows him to compute exactly one of those two values, maintaining the invariant. At the end of the protocol, the receiver has stored ℓ intermediate keys (ℓ being the depth of the tree) which allow to compute all PRF outputs, except one. Transmitting a single additional value allow the sender to reveal

him this value up to an offset β. Due to space constraints, we defer the security analysis of this protocol to Appendix E.

5.2 Semi-Honest Non-Interactive PCG Protocol for Subfield-VOLE Correlations

We now explain how to implement a semi-honest public-key PCG for the subfield-VOLE correlation in the $\mathcal{F}_{\text{PPRF-GGM}}$-hybrid model, by describing a 2-message 2-party semi-honest protocol to distributively execute the procedure $G_{\text{sVOLE}}.\text{Gen}$. The functionality $\mathcal{F}_{\text{Gen}}$ is represented on Figure 5. When $p > 2$, the implementation requires in addition a single (subfield-) reversed vector-OLE on vectors of length t. Reverse vector-OLE can be implemented in two rounds under an appropriate variant of LPN [4] or using linearly homomorphic encryption. We represent the functionality $\mathcal{F}_{\text{rev-VOLE}}$ on Figure 6.

Theorem 5.2. *There exists a 2-message protocol Π_{Gen} which realizes the functionality $\mathcal{F}_{\text{Gen}}(1^\lambda, N, t, p, r)$, with semi-honest security in the $(\mathcal{F}_{\text{PPRF-GGM}}, \mathcal{F}_{\text{rev-VOLE}})$-hybrid model, using t calls to $\mathcal{F}_{\text{PPRF-GGM}}$, a single call to $\mathcal{F}_{\text{rev-VOLE}}(t, p)$, and no further communication. Furthermore, when $p = 2$, the functionality can be implemented directly using t calls to $\mathcal{F}_{\text{PPRF-GGM}}$, and no call to $\mathcal{F}_{\text{rev-VOLE}}$.*

We present the protocol Π_{Gen} in Figure 7. Correctness follows easily by inspection: for $i = 1$ to t, we have $z_i = w_i + c_i = (b_i - \text{PPRF.Eval}(\mathsf{k}_{\text{pprf}}, \alpha_i)) + c_i = x \cdot y_i - \text{PPRF.Eval}(\mathsf{k}_{\text{pprf}}, \alpha_i)$. Security is straightforward. We note that when $p = 2$, since $\vec{y}$ is a weight-t vector, it always hold that $y_i = 1$, hence computing a share of $x \cdot y_i = x$ is trivial and does not require a call to the VOLE functionality.

Implementing $\mathcal{F}_{\text{PPRF-GGM}}$ with the protocol $\Pi.\text{Gen}$ and $\mathcal{F}_{\text{OT}}$ with any 2-round semi-honest OT protocol, this immediately leads to a semi-honest non-interactive PCG protocol $\Pi_{\text{sVOLE}}(\mathbb{F}_q)$ for the subfield-VOLE correlation:

- $\text{R.Gen}(1^\lambda)$: sets pk_{R} to be the first message of $\Pi.\text{Gen}$ and sk_{R} to be the secret state of R.
- $\text{S.Gen}(\mathsf{pk}_{\mathsf{R}})$: sets m_{S} to be the second message of $\Pi.\text{Gen}$ on first message pk_{R}, and sk_{S} to be the sender output in $\Pi.\text{Gen}$.
- $\text{R.Expand}(\mathsf{sk}_{\mathsf{R}}, m_{\mathsf{S}})$: computes the output k_0 of the receiver from the state sk_{R} and the second message m_{S}, and outputs $G_{\text{sVOLE}}.\text{Expand}(0, \mathsf{k}_0)$.
- $\text{S.Expand}(\mathsf{sk}_{\mathsf{S}})$: outputs $G_{\text{sVOLE}}.\text{Expand}(1, \mathsf{sk}_{\mathsf{S}})$.

Corollary 5.3. *Assuming the $(\mathcal{HW}_t, H, \mathbb{F}_q)$-dual-LPN$(n', n)$ assumption, Π_{sVOLE} is a semi-honest non-interactive PCG protocol for subfield-VOLE correlations over an arbitrary extension field $\mathbb{F}_q$ of $\mathbb{F}_2$, which only makes a black-box use of a 1-out-of-2 semi-honest 2-message OT and a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. By making additionally a single black-box use of a 2-message length-t semi-honest reverse VOLE, this can be generalized to arbitrary fields.*

In the above corollary, Π_{sVOLE} makes $t \cdot \lceil \log n' \rceil$ black-box accesses to the 1-out-of-2 semi-honest 2-message OT, $t \cdot n'$ black-box accesses to a length-doubling PRG, and additionally computes one matrix-vector multiplication with H. Regarding communication, the size of pk_{R} is $t \cdot \lceil \log n' \rceil \cdot N_{\mathsf{R}}$ and the size of m_{S} is $t \cdot (\lambda + \lceil \log n' \rceil \cdot N_{\mathsf{S}})$, where N_{R} (resp. N_{S}) denote the receiver communication (resp. the

Protocol $\Pi_{\text{PPRF-GGM}}$:

Parameters: $1^\lambda, \ell, p, r \in \mathbb{N}$. PPRF is the GGM puncturable PRF with domain $\{0,1\}^\ell$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}_{p^r}$, constructed from a length-doubling PRG $G : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$, and a second PRG $G' : \{0,1\}^\lambda \mapsto (\mathbb{F}_{p^r})^2$ used to compute the PRF outputs on the last level of the tree.

Inputs:

- R inputs $\alpha \in \{0,1\}^\ell$.
- S inputs $\beta \in \mathbb{F}_{p^r}$ and a PPRF key $\mathsf{k}_{\text{pprf}} \in \{0,1\}^\lambda$.

Protocol:

(1) R and S execute in parallel ℓ calls to $\mathcal{F}_{\text{OT}}$, where for $i = 1$ to $\ell - 1$:
   - R uses as input the choice bit $\overline{\alpha_i}$;
   - S computes the 2^i partial evaluations at level i of the GGM tree defined by k, denoted $s_0^i, \ldots, s_{2^i-1}^i$ (in left-to-right order) and uses the two OT inputs
   $$t_L^i = \bigoplus_{j \in [0, 2^{i-1})} s_{2j}^i, \quad t_R^i = \bigoplus_{j \in [0, 2^{i-1})} s_{2j+1}^i.$$
   and for the last OT,
   - R uses as input the choice bit $\overline{\alpha_\ell}$;
   - S computes the 2^ℓ evaluations of the GGM tree defined by k, denoted $s_0^\ell, \ldots, s_{2^i-1}^\ell \in (\mathbb{F}_{p^r})^{2^\ell}$ (in left-to-right order) and uses the two OT inputs
   $$t_L^\ell = \sum_{j=0}^{2^{\ell-1}} s_{2j}^\ell, \quad t_R^\ell = \sum_{j=0}^{2^{\ell-1}} s_{2j+1}^\ell.$$

(2) In parallel to the OT calls, S sends $c = \beta - (t_L^\ell + t_R^\ell)$ to R.

Output: R computes its output as follows:

(1) Let t^1 be R's output in the first OT. Define $s_{\overline{\alpha_1}}^1 = t^1$.
(2) For $i = 2, \ldots, \ell - 1$:
   (a) Compute $(s_{2j}^i, s_{2j+1}^i) = G(s_j^{i-1})$, for $j \in [0, \ldots, 2^{i-1}), j \neq \alpha_1 \cdots \alpha_{i-1}$.
   (b) Let t^i be the output from the i-th OT.
   (c) Define $\alpha_i^* = \alpha_1 \cdots \alpha_{i-1}\overline{\alpha_i}$. Compute
   $$s_{\alpha_i^*}^i = t^i \oplus \bigoplus_{\substack{j \in [0, 2^{i-1}), \\ j \neq \alpha_i^*}} s_{2j+\overline{\alpha_i}}^i$$
(3) Compute $(s_{2j}^\ell, s_{2j+1}^\ell) = G'(s_j^{i-1})$, for $j \in [0, \ldots, 2^{\ell-1}), j \neq \alpha_1 \cdots \alpha_{\ell-1}$.
(4) R receives c, and computes
$$t = c + t^\ell + \sum_{j=0, j \neq \alpha}^{2^{\ell-1}} s_{2j+\alpha_\ell}^\ell$$
(5) R outputs the punctured key $\{s_{\alpha_i^*}^i\}_{i \in [\ell]}$, and the final correction value t.

Figure 4: Protocol $\Pi_{\text{PPRF-GGM}}$ for distributing a GGM-based PPRF correlation with semi-honest security in the $\mathcal{F}_{\text{OT}}$-hybrid model

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

PARAMETERS: $1^\lambda, N, t, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N]$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}_{p^r}$.

INPUTS:

- R inputs a weight-t vector $\vec{e} \in \mathbb{F}_p^N$. Let $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$, and $y_i = e_{\alpha_i} \in \mathbb{F}_p^*$.
- S inputs $x \in \mathbb{F}_{p^r}$ and a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.

FUNCTIONALITY:

(1) Compute $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{PPRF.Puncture}(\mathsf{k}_{\mathsf{pprf}}, \mathcal{S})$.
(2) For $i = 1, \ldots, t$, let $z_i \leftarrow x \cdot y_i - \mathsf{PPRF.Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha_i)$
(3) Let $\mathsf{k}_0 \leftarrow (\mathsf{k}^*_{\mathsf{pprf}}, \mathcal{S}, \vec{y}, \{z_i\}_{i\in[t]})$ and $\mathsf{k}_1 \leftarrow (\mathsf{k}_{\mathsf{pprf}}, x)$.
(4) Output k_0 to R and k_1 to S.

Figure 5: Functionality for the Generation Procedure of the Subfield VOLE Generator

Functionality $\mathcal{F}_{\mathsf{rev\text{-}VOLE}}$

PARAMETERS: $t, p, r \in \mathbb{N}$.

INPUT: The sender S inputs a pair $(\vec{b}, x) \in \mathbb{F}_{p^r}^t \times \mathbb{F}_{p^r}$. The receiver R inputs a vector $\vec{y} \in \mathbb{F}_p$.

FUNCTIONALITY: Compute $\vec{c} \leftarrow \vec{y}x - \vec{b}$ and output $\vec{c}$ to R.

Figure 6: Reverse Vector-OLE Functionality over a Field $\mathbb{F}_p$

sender communication) in the underlying OT protocol; over general fields, there is an additional $+M_{\mathsf{R}}(t,q,r)$ term in the size of pk_{R} and $+t \cdot M_{\mathsf{S}}(t,q,r)$ in the size of m_{S}, where $M_{\mathsf{R}}(t,q,r)$ (resp. $M_{\mathsf{S}}(t,q,r)$) denote the receiver communication (resp. the sender communication) in the underlying length-t reverse subfield-VOLE protocol over $\mathbb{F}_{q^r}$.

5.3 Semi-Honest Non-Interactive Secure Computation with Silent Preprocessing

While the non-interactive PCG protocols of the previous section are interesting in their own right, we observe that they satisfy the features outlined in Section 3.2, and therefore lead to 2-round protocols, and even *silent NISC*, for the OT and the VOLE functionalities.

5.3.1 Semi-Honest 2-Round OT with Silent Preprocessing. As observed in [17], a PCG for subfield-VOLE correlation together with a correlation-robust hash function lead to a PCG G_{OT} for the ROT correlation. For completeness, we recall the construction G_{OT} on Figure 15 in Appendix E. Using our distributed generation algorithm (which can be implemented in two rounds given any 2-round OT and 2-round subfield-VOLE), together with the standard protocol for chosen-input OT from ROT, directly leads to a 2-round OT extension protocol, which performs n OTs on s-bit strings with communication $(2s+1) \cdot n + o(n)$ (for any s).

THEOREM 5.4. *Assuming the* $(\mathcal{HW}_t, H, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption,* Π_{OT} *is a* semi-honest 2-round OT extension with silent preprocessing *for generating* n 1*-out-of-*2 *OTs, which makes* $o(n)$ *black-box uses of a 2-round semi-honest* 1*-out-of-*2 *OT, and* $O(n)$ *black-box uses to a length-doubling PRG and an* $\mathbb{F}_p$*-correlation robust hash function.*

Assuming further any 2-round semi-honest reverse VOLE, there is a 2-round OT extension with silent preprocessing for 1*-out-of-*p *OT with comparable costs, using additionnally one black-box execution of a reverse-VOLE on length-*$o(n)$ *inputs.*

Protocol Π_{Gen}

PARAMETERS: $1^\lambda, N = 2^\ell, t, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{p^r}$.

INPUTS:

- R inputs a weight-t vector $\vec{e} \in \mathbb{F}_p^N$. Let $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$, and $y_i = e_{\alpha_i} \in \mathbb{F}_p^*$.
- S inputs $x \in \mathbb{F}_{p^r}$ and a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.

PROTOCOL:

(1) S picks $\vec{b} \xleftarrow{\$} \mathbb{F}_p^t$.
(2) R and S call $\mathcal{F}_{\mathsf{rev\text{-}VOLE}}(t,p)$ on respective inputs $(\vec{y}, (\vec{b}, x))$. R receives an output $\vec{c}$.
(3) For $i = 1$ to t, R and S call $\mathcal{F}_{\mathsf{PPRF\text{-}GGM}}(1^\lambda, \ell, p, r)$ on respective inputs α_i and $(b_i, \mathsf{k}_{\mathsf{pprf}})$. R receives an output $(\mathsf{k}^*_{\mathsf{pprf}}, w_i)$.
(4) For $i = 1$ to t, R computes $z_i \leftarrow w_i + c_i$. R outputs $(\mathsf{k}^*_{\mathsf{pprf}}, \mathcal{S}, \vec{y}, \{z_i\}_{i\in[t]})$ and S outputs $(\mathsf{k}_{\mathsf{pprf}}, x)$.

Figure 7: Protocol for the Generation Procedure of the Subfield VOLE Generator

Assuming further any 2-round semi-honest reverse VOLE, there is a 2-round OT extension with silent preprocessing for 1*-out-of-*p *OT with comparable costs, using additionnally one black-box execution of a reverse-VOLE on length-*$o(n)$ *inputs.*

In the above theorem, Π_{OT} additionally requires the computation of one matrix-vector multiplication with H. It has total communication $(2s+1) \cdot n + o(n)$, where s is the bit-length of the sender messages. We represent the protocol for 2-round OT extension on Figure 8. Due to space constraints, we defer the security analysis of this protocol to Appendix E.

5.3.2 NISC for OT with Silent Preprocessing. Our 2-round OT extension protocol does actually directly give rise to a non-interactive secure computation protocol for the oblivious transfer functionality, with silent preprocessing, as defined in Section 3.2. For the sake of concreteness, we frame our OT extension protocol into the language of NISC with silent preprocessing on Figure 16 of Appendix E.

5.3.3 Semi-Honest NISC for Subfield VOLE. The same derandomization strategy as above directly implies, starting from the non-interactive PCG protocol for subfield-VOLE of Section 5.2, a NISC protocol for subfield VOLE with silent preprocessing, with features comparable to that of the NISC for OT extension. We omit the details.

THEOREM 5.5. *Suppose the* $(\mathcal{HW}_t, H, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds. Then there is a* semi-honest NISC protocol for subfield-VOLE with silent preprocessing *for generating length-*n *subfield-VOLEs over an arbitrary field* $\mathbb{F}_p$*, which uses* $o(n)$ *black-box executions of a 2-message semi-honest* 1*-out-of-*2 *OT,* $O(n)$ *black-box*

Protocol Π_{OT}

PARAMETERS: $1^\lambda, n, N = 2^\ell, t, p, r \in \mathbb{N}$. $H \in \mathbb{F}_p^{N\times n}$. PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{p^r}$. $\mathcal{D}_{t,N}$ is a weight-t error distribution over $\mathbb{F}_p^N$.

INPUTS:

- R inputs n field elements $(s_i)_{i\le n} \in \mathbb{F}_p^n$.
- S inputs n length-p vectors $(\vec{m}_i)_{i\le n}$ where each $\vec{m}_i$ is over $(\{0,1\}^\lambda)^p$.

PROTOCOL:

(1) R picks $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$, and $y_i = e_{\alpha_i} \in \mathbb{F}_p^*$. R computes the first part $\vec{u}$ of $G_{\mathsf{OT}}.\mathsf{Expand}(0, \mathsf{k}_0)$ (note that $\vec{u}$ is computed as $\vec{e}\cdot H$ where $\vec{e}$ depends solely on $\vec{e}$). R sets $\vec{t} \leftarrow \vec{u} + \vec{s}$

(2) S samples $x \xleftarrow{\$} \mathbb{F}_{p^r}$ and $\mathsf{k}_{\mathsf{pprf}} \xleftarrow{\$} \mathsf{PPRF.Gen}(1^\lambda)$. He sets $\mathsf{k}_1 \leftarrow (\mathsf{k}_{\mathsf{pprf}}, x)$ and computes $\{w_{i,j}\}_{i\le n, j\le p} \leftarrow G_{\mathsf{OT}}.\mathsf{Expand}(1, \mathsf{k}_1)$.

(3) R computes and sends to S the first round of Π_{Gen} on input $\vec{e}$, together with $\vec{t}$.

(4) S computes and sends to R the second round of Π_{Gen} on input $(x, \mathsf{k}_{\mathsf{pprf}})$ together with $m'_{i,j} \leftarrow m_{i,j} + w_{i,t_i-j}$ for $i = 1$ to n and $j = 1$ to p; R gets an output k_0.

(5) R computes $(\vec{u}, \vec{v}) \leftarrow G_{\mathsf{OT}}.\mathsf{Expand}(0, \mathsf{k}_0)$ and outputs $(m'_{i,s_i} - v_i)_{i\le n}$.

Figure 8: Two-Round OT Extension

calls to a length-doubling PRG, one black-box call to a 2-message semi-honest reverse VOLE, and additionally computes one matrix-vector multiplication with H. It has total communication $(2s+1)\cdot n + o(n)$.

6 MALICIOUS DISTRIBUTED SETUP

In this section, we present protocols for VOLE, OT extension and NISC with security against *malicious* parties. Our final protocol for OT extension takes place in four rounds, and can be compressed to two rounds via Fiat-Shamir.

We begin in Section 6.1 by formalizing and describing an augmented PPRF primitive with a "malicious key verification" procedure, corresponding to the event of when a selective-failure attack will (or not) be identified. The described selective-failure-only security notion is formalized and achieved for distributed generation of a single PPRF key in Section 6.2, and for t PPRF keys (with consistent x) in Section 6.3. Then, in Section 6.4, we build atop this functionality to obtain a PCG protocol for subfield VOLE with standard *malicious* security. In Appendix F.5 we explain how the PCG protocol for subfield VOLE can be converted into a four round PCG protocol for random 1-out-of-p OT correlation. Finally, in Appendix F.6, we show how to apply the Fiat-Shamir heuristic to compress the protocol down to just 2 rounds, relying on a slightly stronger assumption. To obtain silent NISC for the OT functionality, which implies two-round OT extension on chosen inputs, we use the observation (as in our semi-honest protocol) that the receiver and sender can derandomize their inputs in parallel with their protocol messages. For details and a the protocol for two-round malicious 1-out-of-2 OT extension, we refer to Appendix F.6.

6.1 Puncturable PRF with Malicious Keys

In the following sections we will realize a relaxed form of distributed PPRF setup functionality, where a corrupt sender may choose its own "master key," defining a PRF evaluation that need not coincide with any honest GGM tree, provided that it is consistent with the receiver's punctured point. The consistency check will serve as the "getting caught" predicate in our ideal functionality. In this section, we introduce necessary terminology in order for the consistency check to be formulated.

Roughly, we will have a malicious key space $\mathcal{K}$, such that given a malicious key $K \in \mathcal{K}$ and a subset of values $I \subseteq \mathcal{X}$ in the domain, one can check whether puncturing K at any of the points in the set I yields a consistent output.

For a formal definition and instantiation for the GGM puncturable PRF, we refer to the Appendix, Section F.1.

In order to allow for a consistency check we use the GGM construction with domain $[2N]$ and range $(\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$ (where the former is the range of the left leaves and the latter the range of the right leaves). We will use an output $((\omega, w), \gamma) \in (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$ as follows: The value w will correspond to the actual output of the PPRF. The value γ will be used to ensure consistency *within a single PPRF evaluation.* The value ω will be used to ensure consistency *across t PPRF evaluations.*

6.2 Malicious Setup for Single-Point PPRF

As mentioned in the previous section, in order to achieve malicious security of a single PPRF evaluation, we use the redundancy introduced via the domain extension for checking consistent behaviour, by letting the sender provide a hash of all right leaves of the fully evaluated GGM tree. The idea is that a sender computing the correct hash value (relative to the receivers input α), either behaved honestly, or guessed a set I such that $\alpha \in I$. This is captured in the functionality in Figure 18 in Appendix F.2. The functionality is similar to the semi-honest functionality given in Figure 18, but the adversary is additionally allowed to give a set $I \subseteq [N]$ as guess. If indeed $\alpha \in I$, the sender will successfully finish the protocol and learn some partial information about α (namely, $\alpha \in I$). Otherwise, the functionality will abort.

In order for the right leaves of the GGM tree to fix a *unique* tree, we require the PRG of the last level $G' \colon \{0,1\}^\lambda \to (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$ to satisfy an additional property we call *right-half injectivity*, formally defined in Appendix F.2.

The protocol we present implements the functionality for the PPRF_1, which corresponds to the GGM PPRF, but where evaluation returns a value in $(\mathbb{F}_{p^r})^2$ instead of $(\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$.

We give the protocol for distributed setup of PPRF_1 with security against malicious adversaries in Figure 17 in the appendix, and provide a short description in the following. First, the parties run the semi-honest protocol, such that the receiver holds a key k^* punctured at $\alpha\|0$ and the sender a possibly malicious key K. As the tree is punctured at an even value, both parties can compute all the right leaves of the GGM tree. The sender sends additionally a hash

of all these leaves to the receiver. The receiver checks if this hash is consistent with his view and aborts otherwise.

THEOREM 6.1. *Assuming a black-box access to a PRG* $G : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$*, a right-half injective PRG* $G' : \{0,1\}^\lambda \mapsto (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$*, and a collision resistant hash function* $h\colon \{0,1\}^{\lambda N} \to \{0,1\}^\lambda$*, there exists a 2-party protocol implementing* $\mathcal{F}_{\text{m-PPRF}}$ *(see Fig. 18) for the puncturable PRF* PPRF_1*, with malicious security in the parallel OT-hybrid model, and the following efficiency features. The interaction consists of* ℓ parallel *calls to* $\mathcal{F}_{\text{OT}}$*, and uses additional communication of* $r \log p + \lambda$*. The computational complexity is dominated by* $O(2^\ell)$ *calls each to* G *and* G'*.*

6.3 Malicious Setup of t PPRFs with Consistent Offset

For the VOLE setup with malicious security, we require a protocol for distributed setup of t PPRFs, where the inputs β_j of the sender are *consistent* across all evaluations. By consistent, we mean that each β_j is an additive share of $x \cdot y_j$, where the receiver knows the other share and the noise value $y_j \in \mathbb{F}_p^*$. To this end, we introduce a second consistency check, where the sender has to provide a linear combination of the outputs of each PPRF. We show that a cheating sender will fail this final check, unless he managed to guess part of the receiver's input. This guessing is modelled by the functionality $\mathcal{F}_{\text{m-Gen}}$ (Fig. 20 in Appendix F.3), which is parameterized by a 1-puncturable PRF with verification of malicious keys.

To carry out this check, we exploit the extended range of the PPRF given by the functionality $\mathcal{F}_{\text{m-PPRF}}$. The extra $\mathbb{F}_{p^r}$ element from each evaluation serves to check consistency, by taking a random linear combination of all these outputs (for each PPRF), together with a linear combination of the original outputs, and sending these to the receiver to check. Note that without the extended range, sending a linear combination of PPRF outputs to the receiver would leak the sender's input x; with the extra outputs, however, the sender can use a random value χ which serves to mask x.

Since we sacrifice the extended outputs in the consistency check, the functionality $\mathcal{F}_{\text{m-Gen}}$ which we realize gives us a PPRF with range $\mathbb{F}_{p^r}$, which is defined by simply ignoring the first element output from the one with range $\mathbb{F}_{p^r}^2$.

To create the shares of $x \cdot y_j$, when $p > 2$ we again need a slightly stronger flavor of reverse VOLE, presented in Figure 3 in Appendix F.3. Note that it is not enough for our protocol to instead call the basic reverse VOLE functionality twice, as a receiver providing inconsistent inputs in the two calls, can learn the input x of the sender in the protocol $\Pi_{\text{m-Gen}}$ (Figure 21, Appendix F.3).

For a proof of the following theorem we refer to Appendix F.3.

THEOREM 6.2. *There exists a 4-message 2-party protocol* $\Pi_{\text{m-Gen}}$ *(see Fig. 21 in Section F.2) which securely implements the functionality* $\mathcal{F}_{\text{m-Gen}}(1^\lambda, N, p, r)$ *(see Fig. 20 in Section F.2) for the puncturable PRF* PPRF *in the* $\mathcal{F}_{\text{g-rev-VOLE}^-}$*, parallel* $\mathcal{F}_{\text{m-PPRF}}$*-hybrid model, with malicious security, using* t *parallel calls to* $\mathcal{F}_{\text{m-PPRF}}$*, and only one call to* $\mathcal{F}_{\text{g-rev-VOLE}}$*, and further communication of* $(N + t + 2)r \log p$ *bits. Furthermore, when* $p = 2$*, the functionality can be implemented in the parallel* $\mathcal{F}_{\text{m-PPRF}}$*-hybrid model, using no call to* $\mathcal{F}_{\text{g-rev-VOLE}}$*.*

Note that an additional PRG with range $\mathbb{F}_{p^r}^{N+1}$, the communication can be reduced to just $(t + 1)r \log p + \lambda$ bits.

6.4 4-Round VOLE and OT Setup with Malicious Security

The $\mathcal{F}_{\text{m-Gen}}$ functionality can be immediately used to distribute the setup of the subfield-VOLE PCG from Section 4. To prove this gives secure subfield-VOLE, however, we now need to assume that the dual-LPN assumption remains secure when an adversary is allowed to query (on average) one bit of information on the error vector. This reflects the fact that a malicious sender in $\mathcal{F}_{\text{m-Gen}}$ can try to guess subsets containing the receiver's α_j inputs, which correspond to non-zero coordinates of the error vector. This assumption with leakage is essentially the same as an assumption recently used for maliciously secure MPC based on syndrome decoding [34]. For a formal definition we refer to Definition F.4 in Appendix F.4.

THEOREM 6.3. *Let* PPRF *be a* t*-puncturable PRF, and suppose that* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(N, n)$ *with static leakage holds. The protocol in Fig. 23 securely realizes the functionality* $\mathcal{F}_{\text{sVOLE}}$*.*

COROLLARY 6.4. *Suppose that* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(N, n)$ *with static leakage holds, where* $N = O(n)$ *and* $t = o(n/(\lambda \log n)$*. Then there exists a 4-message, maliciously secure PCG protocol for the subfield VOLE correlation, which makes* $o(n)$ *parallel calls to an oblivious transfer functionality, with communication complexity* $o(n)$ *bits.*

7 IMPLEMENTATION

7.1 Instantiating the Code and Parameters

The most costly part of our implementation is the syndrome computation with the matrix H used in the dual-LPN assumption. We optimize this by instantiating H using the parity-check matrix of a quasi-cyclic code. Multiplication by H can then be done with polynomial arithmetic in $\mathbb{Z}_2[X]/(X^n - 1)$, for which we use the library `bitpolymul` [21]. Another optimization that improves efficiency and reduces the seed size is to use a regular error distribution, where the error vector $\vec{e} \in \mathbb{F}_2^N$ is the concatenation of t random unit vectors, each of length N/t. To choose the code parameters N, n and the error weight t, we analyze security against the best known attacks, additionally accounting for a $\sqrt{N}$ speedup that can be obtained from the DOOM attack [57] when using quasi-cyclic codes. As also observed in [35], we are not aware of any attacks that exploit regular errors and perform significantly better than usual.

In Appendix A.2 and Table 1 of the appendix, we provide more details on selecting parameters and describe some further optimizations for the syndrome computation.

7.2 Results

We implement our semi-honest and malicious secure protocols and report their performance in several different settings. The source code can be found at github.com/osu-crypto/libOTe. The benchmark was performed on a single AWS c4.4xLarge instance with network latency artificially limited to emulate a LAN or WAN settings. Specifically, we consider a LAN setting with bandwidth of 10Gbps and 0ms latency and two WAN settings with 100, 10 Mbps & 40ms one-way latency. Our implementation will be freely available. We compare with the semi-honest OT extension protocol of Ishai et al. [37] (IKNP) and the malicious secure protocol of [41] (KOS)

				LAN (10Gbps) times				WAN (100Mbps) times				WAN (10Mbps) times			
				n				n				n			
Protocol	Base type	λ	τ	10^7	10^6	10^5	10^4	10^7	10^6	10^5	10^4	10^7	10^6	10^5	10^4
This (SH)	hybrid	128	4	2,441	208	**76**	**67**	**2,726**	**513**	**422**	**425**	**2,756**	**518**	**454**	**422**
IKNP	base	128	4	**268**	**125**	94	91	13,728	1,850	493	459	128,954	13,332	1,756	445
This (SH)	hybrid	128	1	7,990	533	130	100	**8,252**	**808**	**451**	422	**8,291**	**815**	**467**	**422**
IKNP	base	128	1	**573**	**157**	**108**	**98**	15,622	2,030	613	**341**	129,011	13,285	1,672	429
This (Mal)	hybrid	128	4	2659	280	**84**	**78**	**2872**	**479**	**457**	**424**	**2846**	**515**	**438**	**422**
KOS	base	128	4	**333**	**121**	110	111	13722	1933	589	426	129052	13391	1804	536
This (Mal)	hybrid	128	1	8765	584	141	**104**	**9055**	**828**	**460**	**423**	**8929**	**831**	**467**	**433**
KOS	base	128	1	**674**	**170**	**113**	106	15741	2088	702	433	129771	13389	1772	518

Figure 9: The running time in milliseconds of our implementation compared to [8] in both the LAN (0ms latency) and WAN (40ms one-way latency) settings, with security parameter $\lambda = 128$. λ is the computational security parameter. We set the scaling N/n to 2. τ denotes the number of threads. Hybrid refers to doing 128 base OTs followed by IKNP to derive the total required base OTs.

		Total Comm. (bytes)				Comm./OT (bits)			
		n				n			
Protocol	Base type	10^7	10^6	10^5	10^4	10^7	10^6	10^5	10^4
This (SH/Mal)	hybrid	126,658	98,754	83,394	57,806	0.101	0.790	6.672	46.245
IKNP/KOS	base	160,056,360	16,011,518	1,655,784	168,186	128.045	128.092	132.463	134.549

Figure 10: The communication overhead of our implementation compared to [37, 41], with $N/n = 2$ and $\lambda = 4$. See Figure 9.

as implemented by a state-of-the-art library. Both our implementations and that of [37, 41] use the same three round malicious secure base OT protocol of Naor & Pinkas[51]. We note that our protocols can be composed with a two round base OT protocol to give a two round OT extension. In the WAN setting this optimization would reduce the running times by approximately 40ms for all protocols.

The functionality we realize is to produce $n \in \{10^4, 10^5, 10^6, 10^7\}$ uniformly random OTs of length 128 bits. One distinction between our protocol and [37, 41] is that the choice bits of the receiver are uniformly chosen by our protocol, while [37, 41] allows the receiver to specify them. These random OTs can then be de-randomized with additional communication.

Figure 9 contains the running time of our protocol. A fuller table, with alternative choices of parameters (security parameter λ, scaling parameter N/n, method for computing the base OTs) is available on Figure 11 of Appendix A.1. The primary takeaway is that both of our protocols achieve extremely low communication while the total running time remains competitive with or superior to KOS and IKNP. We report running times with each party having 1 or 4 threads, along with a background IO thread. In the LAN setting with sub-millisecond latency & 10Gbps we observe that the IKNP and KOS protocols achieve significant performance, requiring just 0.26 or 0.33 seconds to compute 10 million OTs with a single thread. While the computational cost of IKNP and KOS does outperform our implementation by roughly one order of magnitude, it also requires between 1000 and 2000 times more communication. This difference means that for more realistic network settings, such as 100Mbps, our implementation achieves a faster running time. With 4 threads and a limit of 100Mbps our implementation is up to 5 times faster (counting total running time, including both local computation and communication costs) and remains faster even for small n where our communication overheads are asymptotically closer together.

For the constrained setting of 10Mbps our protocol truly stands out with a 47 times speedup compared to IKNP with $n = 10^7$ and $t = 4$. We see a similar 46 times speedup in the malicious setting compared to KOS. Moreover, when comparing between the across the different network settings our protocol incurs minimal to no perform impact from decreasing bandwidth. For instance, with a 10Gbps connection our semi-honest protocol processes $n = 10^7$ OTs in 2.4 seconds while with 1000 times less bandwidth the protocol still just requires 2.8 seconds.

This scalability is explained in Figure 10 which contains the communication overhead of our protocol. A fuller table, with alternative choices of parameters (security parameter λ, scaling parameter N/n, method for computing the base OTs) is available on Figure 12 of Appendix A.1. We parameterize our protocols by the desired security level $\lambda \in \{80, 128\}$ and a tunable parameter $s = N/n$. The latter controls a trade-off between the number of PPRF evaluations and length of the resulting vectors. To maintain security level of λ bits, increasing s results in fewer PPRF evaluations and less communication. However, it also increases the computational overhead. Our smallest running times were achieved with $s = 2$. However, we also consider $s = 4$ which decreases our total communication from 126KB to 80KB for $n = 10^7$. In contrast, the IKNP protocol requires 160MB for the same security level. This represents as much as a 2000 times reduction in communication. This low communication overhead results in our protocol requiring as little as 0.038 bits per OT for $n = 10^7$ and $\lambda = 80$. In our worst case of $n = 10^4$ our protocol still requires between 3 and 6 times less communication than IKNP. Another compelling property of our protocol is that

we incur near constant additive communication overhead when comparing our malicious and semi-honest protocols.

REFERENCES

[1] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. 2014. Non-Interactive Secure Computation Based on Cut-and-Choose. In *EUROCRYPT 2014 (LNCS)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.), Vol. 8441. Springer, Heidelberg, 387–404. https://doi.org/10.1007/978-3-642-55220-5_22

[2] Carlos Aguilar, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. 2016. Efficient Encryption from Random Quasi-Cyclic Codes. Cryptology ePrint Archive, Report 2016/1194. (2016). http://eprint.iacr.org/2016/1194.

[3] Michael Alekhnovich. 2003. More on Average Case vs Approximation Complexity. In *44th FOCS*. IEEE Computer Society Press, 298–307. https://doi.org/10.1109/SFCS.2003.1238204

[4] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. 2017. Secure Arithmetic Computation with Constant Computational Overhead. In *CRYPTO 2017, Part I (LNCS)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10401. Springer, Heidelberg, 223–254. https://doi.org/10.1007/978-3-319-63688-7_8

[5] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. 2009. Cryptography with Constant Input Locality. *Journal of Cryptology* 22, 4 (Oct. 2009), 429–469. https://doi.org/10.1007/s00145-009-9039-0

[6] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. 2017. Bike: Bit flipping key encapsulation. (2017).

[7] Sanjeev Arora and Rong Ge. 2011. New Algorithms for Learning in Presence of Errors. In *ICALP 2011, Part I (LNCS)*, Luca Aceto, Monika Henzinger, and Jiri Sgall (Eds.), Vol. 6755. Springer, Heidelberg, 403–415. https://doi.org/10.1007/978-3-642-22006-7_34

[8] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 535–548. https://doi.org/10.1145/2508859.2516738

[9] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. 420–432. https://doi.org/10.1007/3-540-46766-1_34

[10] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO'95 (LNCS)*, Don Coppersmith (Ed.), Vol. 963. Springer, Heidelberg, 97–109. https://doi.org/10.1007/3-540-44750-4_8

[11] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. 2012. Decoding Random Binary Linear Codes in $2^{n/20}$: How 1 + 1 = 0 Improves Information Set Decoding. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 520–536. https://doi.org/10.1007/978-3-642-29011-4_31

[12] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011 (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 169–188. https://doi.org/10.1007/978-3-642-20465-4_11

[13] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. 1993. Cryptographic Primitives Based on Hard Learning Problems. In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*. 278–291. https://doi.org/10.1007/3-540-48329-2_24

[14] Avrim Blum, Adam Kalai, and Hal Wasserman. 2000. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*. ACM Press, 435–440. https://doi.org/10.1145/335305.335355

[15] Dan Boneh and Brent Waters. 2013. Constrained Pseudorandom Functions and Their Applications. In *ASIACRYPT 2013, Part II (LNCS)*, Kazue Sako and Palash Sarkar (Eds.), Vol. 8270. Springer, Heidelberg, 280–300. https://doi.org/10.1007/978-3-642-42045-0_15

[16] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing Vector OLE. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 896–912. https://doi.org/10.1145/3243734.3243868

[17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO 2019, Part III (LNCS)*, Alexandra Boldyreva and Daniele Micciancio (Eds.), Vol. 11694. Springer, Heidelberg, 489–518. https://doi.org/10.1007/978-3-030-26954-8_16

[18] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 2014. Functional Signatures and Pseudorandom Functions. In *PKC 2014 (LNCS)*, Hugo Krawczyk (Ed.), Vol. 8383. Springer, Heidelberg, 501–519. https://doi.org/10.1007/978-3-642-54631-0_29

[19] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology* 13, 1 (Jan. 2000), 143–202. https://doi.org/10.1007/s001459910006

[20] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, 136–145. https://doi.org/10.1109/SFCS.2001.959888

[21] Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang. 2018. Multiplying boolean Polynomials with Frobenius Partitions in Additive Fast Fourier Transform. *CoRR* abs/1803.11301 (2018).

[22] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012 (LNCS)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, Heidelberg, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[23] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. 2017. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *NDSS 2017*. The Internet Society.

[24] Jack Doerner and abhi shelat. 2017. Scaling ORAM for Secure Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 523–535. https://doi.org/10.1145/3133956.3133967

[25] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. 2017. TinyOLE: Efficient Actively Secure Two-Party Computation from Oblivious Linear Function Evaluation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2263–2276. https://doi.org/10.1145/3133956.3134024

[26] Erez Druk and Yuval Ishai. 2014. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS 2014*, Moni Naor (Ed.). ACM, 169–182. https://doi.org/10.1145/2554797.2554815

[27] Andre Esser, Robert Kübler, and Alexander May. 2017. LPN Decoded. In *CRYPTO 2017, Part II (LNCS)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10402. Springer, Heidelberg, 486–514. https://doi.org/10.1007/978-3-319-63715-0_17

[28] Sanjam Garg, Mohammad Mahmoody, Daniel Masny, and Izaak Meckler. 2018. On the Round Complexity of OT Extension. In *CRYPTO 2018, Part III (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10993. Springer, Heidelberg, 545–574. https://doi.org/10.1007/978-3-319-96878-0_19

[29] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. 2017. Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead. In *ASIACRYPT 2017, Part I (LNCS)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.), Vol. 10624. Springer, Heidelberg, 629–659. https://doi.org/10.1007/978-3-319-70694-8_22

[30] Oded Goldreich. 2004. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA.

[31] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to Construct Random Functions. *Journal of the ACM* 33, 4 (Oct. 1986), 792–807.

[32] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420

[33] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. 2019. Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers. Cryptology ePrint Archive, Report 2019/074. (2019). https://eprint.iacr.org/2019/074.

[34] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2018. Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT). In *ASIACRYPT 2018, Part III (LNCS)*, Thomas Peyrin and Steven Galbraith (Eds.), Vol. 11274. Springer, Heidelberg, 86–117. https://doi.org/10.1007/978-3-030-03332-3_4

[35] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2018. TinyKeys: A New Approach to Efficient Multi-Party Computation. In *CRYPTO 2018, Part III (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10993. Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-319-96878-0_1

[36] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS 2012*. The Internet Society.

[37] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, Heidelberg, 145–161. https://doi.org/10.1007/978-3-540-45146-4_9

[38] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. 2011. Efficient Non-interactive Secure Computation. In *EUROCRYPT 2011 (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 406–425. https://doi.org/10.1007/978-3-642-20465-4_23

[39] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2009. Secure Arithmetic Computation with No Honest Majority. In *TCC 2009 (LNCS)*, Omer Reingold (Ed.), Vol. 5444. Springer, Heidelberg, 294–314. https://doi.org/10.1007/978-3-642-00457-5_18

[40] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. 2018. Optimizing Authenticated Garbling for Faster Secure Two-Party Computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III.*

365–391. https://doi.org/10.1007/978-3-319-96878-0_13
[41] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2015. Actively Secure OT Extension with Optimal Overhead. In *CRYPTO 2015, Part I (LNCS)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.), Vol. 9215. Springer, Heidelberg, 724–741. https://doi.org/10.1007/978-3-662-47989-6_35
[42] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. 2013. Delegatable pseudorandom functions and applications. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 669–684. https://doi.org/10.1145/2508859.2516668
[43] Joe Kilian. 1988. Founding Cryptography on Oblivious Transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. 20–31. https://doi.org/10.1145/62212.62215
[44] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT Extension for Transferring Short Secrets. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, 54–70. https://doi.org/10.1007/978-3-642-40084-1_4
[45] Zhen Liu and Yanbin Pan. 2019. NIST Official Comment – HQC. NIST Post-Quantum Cryptography Project. (2019). https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/HQC-official-comment.pdf.
[46] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. 2016. Squaring Attacks on McEliece Public-key Cryptosystems Using Quasi-cyclic Codes of Even Dimension. *Des. Codes Cryptography* 80, 2 (Aug. 2016), 359–377. https://doi.org/10.1007/s10623-015-0099-x
[47] Jin Lu and José MF Moura. 2010. Linear time encoding of LDPC codes. *IEEE Transactions on Information Theory* 56, 1 (2010), 233–249.
[48] Vadim Lyubashevsky. 2005. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, randomization and combinatorial optimization. Algorithms and techniques*. Springer, 378–389.
[49] Carlos Aguilar Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. 2018. Efficient Encryption From Random Quasi-Cyclic Codes. *IEEE Trans. Information Theory* 64, 5 (2018), 3927–3943. https://doi.org/10.1109/TIT.2018.2804444
[50] Payman Mohassel and Mike Rosulek. 2017. Non-interactive Secure 2PC in the Offline/Online and Batch Settings. In *EUROCRYPT 2017, Part III (LNCS)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.), Vol. 10212. Springer, Heidelberg, 425–455. https://doi.org/10.1007/978-3-319-56617-7_15
[51] Moni Naor and Benny Pinkas. 2005. Computationally Secure Oblivious Transfer. *Journal of Cryptology* 18, 1 (Jan. 2005), 1–35. https://doi.org/10.1007/s00145-004-0102-6
[52] Moni Naor and Benny Pinkas. 2006. Oblivious Polynomial Evaluation. *SIAM J. Comput.* 35, 5 (2006), 1254–1281.
[53] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO 2008 (LNCS)*, David Wagner (Ed.), Vol. 5157. Springer, Heidelberg, 554–571. https://doi.org/10.1007/978-3-540-85174-5_31
[54] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security 2015*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 515–530.
[55] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-based PSI with Linear Communication. EUROCRYPT 2019. https://eprint.iacr.org/2019/241.
[56] Eugene Prange. 1962. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory* 8, 5 (1962), 5–9.
[57] Nicolas Sendrier. 2011. Decoding One Out of Many. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, Bo-Yin Yang (Ed.). Springer, Heidelberg, 51–67. https://doi.org/10.1007/978-3-642-25405-5_4
[58] Rodolfo Canto Torres and Nicolas Sendrier. 2016. Analysis of Information Set Decoding for a Sub-linear Error Weight. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, Tsuyoshi Takagi (Ed.). Springer, Heidelberg, 144–161. https://doi.org/10.1007/978-3-319-29360-8_10
[59] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 21–37. https://doi.org/10.1145/3133956.3134053
[60] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 39–56. https://doi.org/10.1145/3133956.3133979
[61] Lior Zichron. 2017. Locally Computable Arithmetic Pseudorandom Generators. Master's thesis, School of Electrical Engineering, Tel Aviv University. (2017). http://www.eng.tau.ac.il/~bennyap/pubs/Zichron.pdf

Table 1: Bit security of dual-LPN, counted as the minimum logarithm of the number of arithmetic operations for each of the ISD attack [11, 56], the low-weight parity check attack [61] and the Gaussian elimination attack [27]. The attacks take into account a speedup from the DOOM attack [57] which is enabled by our use of quasi-cyclic codes.

n	t	N/n	κ	n	t	N/n	κ
10^4	73	2	80	10^4	126	2	128
10^5	72	2	80	10^5	120	2	128
10^6	70	2	80	10^6	118	2	128
10^7	68	2	80	10^7	116	2	128
10^4	37	4	80	10^4	80	4	128
10^5	36	4	80	10^5	72	4	128
10^6	35	4	80	10^6	63	4	128
10^7	34	4	80	10^7	54	4	128

A DETAILS ON IMPLEMENTATION

A.1 Detailed Performance Figures

In this section, we provide more extensive numbers regarding the running time and communication complexity of our implementation, with alternative choices of parameters (security parameter λ, scaling parameter N/n, method for computing the base OTs). Figure 11 contains the running time of our protocol. Figure 12 contains the communication overhead of our protocol.

A.2 Quasi-Cyclic Codes and Parameters

We instantiate the matrix H in the dual-LPN assumption with quasi-cyclic codes. Specifically, let $H' \in \mathbb{F}_2^{N \times n}$ is the parity-check matrix of a random, quasi-cyclic code in systematic form. Writing $N = s \cdot n$, for the cases $s = 2$ and $s = 4$ which we use, these parity-check matrices are respectively defined by

$$H'_2 = \begin{pmatrix} I_n & \mathrm{rot}(h) \end{pmatrix}^\top, \quad H'_4 = \begin{pmatrix} I_n & \mathrm{rot}(h_1) & \mathrm{rot}(h_2) & \mathrm{rot}(h_3) \end{pmatrix}^\top$$

where I_n is the $n \times n$ identity, and $\mathrm{rot}(h)$ is the circulant matrix consisting of all n rotations of the vector $h \in \mathbb{F}_2^n$. Note that multiplication of a vector with $\mathrm{rot}(h)$ is equivalent to a polynomial multiplication in $\mathbb{Z}_2[X]/(X^n - 1)$).

We then define the matrix H to be H' with its final row removed (see *Security* below). For the noise distribution, to improve the efficiency of the puncturable PRF we use a *regular* error vector $\vec{e} \in \mathbb{F}_2^N$, which is the concatenation of t random unit vectors, each of length N/t.

Fast quasi-cyclic encoding. To efficiently implement multiplication by H, we used the library `bitpolymul` [21] for fast multiplication in $\mathbb{Z}_2[X]$. Multiplying two degree n polynomials has complexity $\tilde{O}(n)$ using additive fast Fourier transforms, with an algorithm following the standard $\mathsf{FFT} \rightarrow \mathsf{PointwiseMult} \rightarrow \mathsf{FFT}^{-1}$ structure. We optimize this by preprocessing $\mathsf{FFT}(h_i)$, since the h_i values are fixed, and postponing FFT^{-1} until after summing up the s terms in the multiplication. This reduces computation by around 30–50%.

Security. Recall that for our dual-LPN variant, we require that given H, $\vec{e} \cdot H$ is indistinguishable from a uniform vector, where $\vec{e}$ is a weight-t, regular error vector. The reason we truncated the

Protocol	Base type	λ	s	t	LAN (10Gbps) times n 10^7	10^6	10^5	10^4	WAN (100Mbps) times n 10^7	10^6	10^5	10^4	WAN (10Mbps) times n 10^7	10^6	10^5	10^4
This (SH)	base	80	4	4	4,507	548	301	227	4,721	780	487	403	4,662	815	543	463
	hybrid	80	4	4	4,261	344	118	65	4,598	616	366	337	4,517	600	375	346
	base	128	4	4	4,861	766	421	357	4,932	938	616	539	5,018	929	646	516
	hybrid	128	4	4	4,233	276	85	90	4,595	551	391	337	4,615	552	408	352
	base	128	2	4	3,373	975	634	528	3,675	1,221	875	687	3,603	1,188	851	747
	hybrid	128	2	4	2,441	208	**76**	**67**	**2,726**	**513**	**422**	**425**	**2,756**	**518**	**454**	**422**
IKNP	base	128	-	4	**268**	**125**	94	91	13,728	1,850	493	459	128,954	13,332	1,756	445
This (SH)	base	80	4	1	14,127	1,204	479	305	14,467	1,408	682	505	14,385	1,416	716	534
	hybrid	80	4	1	13,772	812	164	102	13,987	1,067	406	344	13,962	1,060	407	353
	base	128	4	1	14,701	1,445	678	480	15,079	1,642	886	681	14,994	1,649	885	685
	hybrid	128	4	1	13,996	787	163	101	14,344	1,061	474	346	14,156	1,056	471	361
	base	128	2	1	9,414	1,747	1,008	761	9,694	1,973	1,220	964	9,750	1,980	1,226	980
	hybrid	128	2	1	7,990	533	130	100	**8,252**	**808**	**451**	422	**8,291**	**815**	**467**	**422**
IKNP	base	128	-	1	**573**	**157**	**108**	**98**	15,622	2,030	613	**341**	129,011	13,285	1,672	429
This (Mal)	base	128	4	4	5286	879	463	358	5589	1127	787	670	5624	1123	801	670
	hybrid	128	4	4	5030	344	116	69	5141	622	387	339	5292	699	372	354
	base	128	2	4	3674	1018	643	528	3897	1252	891	726	3836	1217	858	777
	hybrid	128	2	4	2659	280	**84**	**78**	**2872**	**479**	**457**	**424**	**2846**	**515**	**438**	**422**
KOS	base	128	-	4	**333**	**121**	110	111	13722	1933	589	426	129052	13391	1804	536
This (Mal)	base	128	4	1	16096	1632	707	490	16499	1947	1033	811	16616	1958	1045	813
	hybrid	128	4	1	15656	968	185	110	15999	1205	489	426	15889	1207	490	426
	base	128	2	1	10475	1833	1028	773	10585	2026	1278	1051	10449	2033	1286	1048
	hybrid	128	2	1	8765	584	141	**104**	**9055**	**828**	**460**	**423**	**8929**	**831**	**467**	**433**
KOS	base	128	-	1	**674**	**170**	**113**	106	15741	2088	702	433	129771	13389	1772	518

Figure 11: The running time in milliseconds of our implementation compared to [8] in both the LAN (0ms latency) and WAN (40ms one-way latency) settings. λ is the computational security parameter. $s = N/n$ denotes the scaling parameter such that the PPRF output strings are of length N. t denotes the number of threads. Hybrid refers to doing 128 base OTs followed by IKNP to derive the total required base OTs.

Protocol	Base type	λ	s	Total Comm. (bytes) n 10^7	10^6	10^5	10^4	Comm./OT (bits) n 10^7	10^6	10^5	10^4
This (SH/Mal)	base	80	4	53,478	45,678	37,878	27,478	0.043	0.365	3.030	21.982
	hybrid	80	4	**47,690**	**43,850**	**40,010**	**34,890**	**0.038**	**0.351**	**3.201**	**27.912**
	base	128	4	85,482	68,842	56,362	43,882	0.068	0.551	4.509	35.106
	hybrid	128	4	**80,238**	**55,662**	**49,518**	**43,374**	**0.064**	**0.445**	**3.961**	**34.699**
	base	80	2	91,470	72,750	58,710	44,418	0.073	0.582	4.697	35.534
	hybrid	80	2	83,322	74,106	50,810	43,910	0.067	0.593	4.065	35.128
	base	128	2	144,558	121,158	89,958	70,986	0.116	0.969	7.197	56.789
	hybrid	128	2	126,658	98,754	83,394	57,806	0.101	0.790	6.672	46.245
IKNP/KOS	base	128	-	160,056,360	16,011,518	1,655,784	168,186	128.045	128.092	132.463	134.549

Figure 12: The communication overhead of our implementation compared to [8]. λ stands for the computational security parameter. See Figure 9.

parity-check matrix H' to form H, is that with quasi-cyclic codes the parity bit of $\vec{e} \cdot H'$ only depends on H and t, so there is a trivial distinguisher [45] which truncating avoids. We also require n to be prime to avoid attacks expoiting the quasi-cyclic structure [46]. We are not aware of any specific attacks exploiting a regular error distribution.

Note that quasi-cyclic codes have been used to construct optimized variants of the LPN-based cryptosystem of Alekhnovich and the code-based cryptosystem of McEliece [2, 49], including several candidates in the ongoing NIST standardization process. Our assumption seems more conservative than these schemes, which need to embed a trapdoor into H that allows efficient decoding.

Choosing Parameters. We evaluate the concrete security of dual-LPN for various parameters (n, N, t), calculating the minimal number of noisy coordinates t such that dual-LPN with dimension

n, number of samples N, and noise rate t/N requires 2^κ arithmetic operations to be broken using state-of-the-art attacks, for $\kappa \in \{80, 128\}$. The main attacks on LPN which apply in our setting (where the number of samples N is strongly restricted and the noise rate t/N is very low) are the low-weight parity check attack [61], the Gaussian elimination attack and its variants [27], and information set decoding (ISD) [56] and its variants, especially BJMM [11]. We evaluated the concrete resistance of our LPN instances against all these attacks. For ISD [11], we relied on the analysis of [58] and of the NIST candidate BIKE [6, Section 5.2], which identify the BJMM attack as the most efficient, and provide a closed formula. Since we rely on quasi-cyclic codes to improve the computational efficiency, we also take into account the effect of the DOOM (Decoding One Out of Many) attack [57] which provides a $\sqrt{N}$ computational speedup against variants of LPN relying on quasi-cyclic codes. The results are summarized in Table 1.

Alternative Codes. Our choice of quasi-cyclic codes over alternative fast codes is mainly motivated by the fact that they are well studied, and fast implementations are available. However, as discussed in [16], we note that alternatives such as Druk-Ishai codes [26] or LDPC codes [3, 47] may be better. Both of these would allow for a *linear time* syndrome computation (instead of quasilinear), with small constants (less than $3d$ in the case of LDPC codes, where d is the row-weight of the sparse parity-check matrix). Moreover, these codes are not sensitive to the DOOM attack [57], and might therefore provide stronger resistance to standard attacks on LPN. Therefore, using these codes would likely allow to improve noticeably the efficiency of our implementation, indicating that there is a lot of room for improvement; we leave this to future work.

B PRELIMINARIES ON LPN

We provide in this section a more formal and thorough introduction to the LPN assumption and its security.

Definition B.1 (LPN). Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{k,q}(\mathcal{R})\}_{k,q\in\mathbb{N}}$ denote a family of distributions over a ring $\mathcal{R}$, such that for any $k, q \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{k,q}(\mathcal{R})) \subseteq \mathcal{R}^q$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, q, \mathcal{R})$ outputs a matrix $A \in \mathcal{R}^{k\times q}$. For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-LPN$(k, q)$ assumption states that

$$\{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}_{k,q}(\mathcal{R}), \vec{s} \xleftarrow{\$} \mathbb{F}^k, \vec{b} \leftarrow \vec{s} \cdot A + \vec{e}\} \\ \overset{c}{\approx} \{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^q\}$$

Here and in the following, all parameters are functions of the security parameter λ and computational indistinguishability is defined with respect to λ. When $\mathcal{R} = \mathbb{F}_2$ and $\mathcal{D}$ is the Bernoulli distribution over $\mathbb{F}_2^q$, where each coordinate is 1 with probability r and 0 otherwise, this corresponds to the standard binary LPN assumption. Note that the search LPN problem, of finding the vector can be reduced to the decisional LPN assumption as defined above above when the code generator $\mathbf{C}$ outputs a uniform matrix A [5, 13]. However, this is less relevant for us as we are mainly interested in efficient variants with more structured codes. See [26] for further discussion of search-to-decision reductions in the general case.

B.0.1 Example: LPN with Fixed Weight Noise. For a finite field $\mathbb{F}$, we denote by $\mathcal{HW}_r(\mathbb{F})$ the distribution of uniform, weight r vectors over $\mathbb{F}$; that is, a sample from $\mathcal{HW}_r(\mathbb{F})$ is a uniformly random nonzero field element in r random positions, and zero elsewhere. The $(\mathsf{Ber}_r(\mathbb{F})^q, \mathbf{C}, \mathbb{F}) - \mathsf{LPN}(k, q)$ assumption corresponds to the standard (non-binary, fixed-weight) LPN assumption over a field $\mathbb{F}$ with code generator $\mathbf{C}$, dimension k, number of samples (or block length) q, and noise rate r.

When the block length q and noise rate r are such that k random coordinates will be all noiseless with non-negligible probability (e.g., when r is constant and $q = \Omega(k^2)$), LPN can be broken via Gaussian elimination (cf. [7]). This attack does not apply to our constructions, which typically have $q = O(k)$.

Definition B.2 (dual LPN). Let $\mathcal{D}(\mathcal{R})$ and $\mathbf{C}$ be as in Definition B.1, $n, N \in \mathbb{N}$ with $N > n$, and define $\mathbf{C}^\perp(N, n, \mathcal{R}) = \{B \in \mathcal{R}^{N\times n} : A \cdot B = 0, A \in \mathbf{C}(N - n, N, \mathcal{R}), \mathsf{rank}(B) = n\}$.

For $n = n(\lambda)$, $N = N(\lambda)$ and $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-dual-LPN$(N, n)$ assumption states that

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}^\perp(N, n, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}(\mathcal{R}), \vec{b} \leftarrow \vec{e} \cdot H\} \\ \overset{c}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}^\perp(N, n, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^n\}$$

We will slightly abuse our notations by omitting to explicitely mention the code $\mathbf{C}$ and writing $(\mathcal{D}, H, \mathcal{R})$-dual-LPN$(N, n)$ for above dual-LPN assumption with a matrix $H \in \mathbf{C}^\perp(N, n, \mathcal{R})$.

The search version of the dual LPN problem is also known as syndrome decoding. The decision version defined above is equivalent to primal variant of LPN from Definition B.1 with dimension $N - n$ and number of samples N. This follows from the simple fact that $(\vec{s} \cdot A + \vec{e}) \cdot H = \vec{s} \cdot A \cdot H + \vec{e} \cdot H = \vec{e} \cdot H$, when H is the parity-check matrix of A.

B.0.2 Attacks on LPN. We recall here the main attacks on LPN, following the analysis of [16]. We refer the reader to [27] for a more comprehensive overview. We assume that $\mathcal{D}$ is a noise distribution with Hamming weight bounded by some integer t.

- **Gaussian elimination.** The most natural attack on LPN recovers $\vec{s}$ from $\vec{b} = \vec{s} \cdot A + \vec{e}$ by guessing n non-noisy coordinates of $\vec{b}$, and inverting the corresponding subsystem to verify whether the guess was correct. This approach recovers $\vec{s}$ in time at least $(1/(1-r))^n$ using at least $O(n/r)$ samples ($r = t/N$). For low-noise LPN, with noise rate $1/n^c$ for some constant $c \geq 1/2$, this translates to a bound on attacks of $O(e^{n^{1-c}})$ time using $O(n^{1+c})$ samples.
- **Information Set Decoding (ISD) [56].** Breaking LPN is equivalent to solving its dual variant, which can be interpreted as the task of decoding a random linear code from its syndrome. The best algorithms for this task are improvements of Prange's ISD algorithm, which attempts to find a size-t subset of the rows of B (the parity-check matrix of the code) that spans $\vec{e} \cdot B$, where $t = rN$ is the number of noisy coordinates. The state of the art variant of Prange's information set decoding attack is the BJMM attack [11], which was analyzed in [58], and in the NIST candidate

BIKE [6, Section 5.2], which also take into account the effect of the DOOM attack [57] which applies to the specific case of LPN with quasi-cyclic codes.

- **The BKW algorithm [14].** This algorithm is a variant of Gaussian elimination which achieves subexponential complexity even for high-noise LPN (e.g. constant noise rate), but requires a subexponential number of samples: the attack solves LPN over $\mathbb{F}_2$ in time $2^{O(n/\log(n/r))}$ using $2^{O(n/\log(n/r))}$ samples.
- **Combinations of the above [27].** The authors of [27] conducted an extended study of the security of LPN, and described combinations and refinements of the previous three attacks (called the *well-pooled Gauss attack*, the *hybrid attack*, and the *well-pooled MMT attack*). All these attacks achieve subexponential time complexity, but require as many sample as their time complexity.
- **Scaled-down BKW [48].** This algorithm is a variant of the BKW algorithm, tailored to LPN with polynomially-many samples. It solves LPN in time $2^{O(n/\log\log(n/r))}$, using $n^{1+\varepsilon}$ samples (for any constant $\varepsilon > 0$) and has worse performance in time and number of samples for larger fields.
- **Low-Weight Parity Check [61].** Eventually, all the previous attacks recover the secret $\vec{s}$. A more efficient attack (by a polynomial factor) can be used if one simply wants to distinguish $\vec{b} = \vec{s} \cdot A + \vec{e}$ from random: by the singleton bound, the minimal distance of the dual code of $\mathbf{C}$ is at most $n + 1$, hence there must be a parity-check equation for $\mathbf{C}$ of weight $n + 1$. Then, if $\vec{b}$ is random, it passes the check with probability at most $1/|\mathbb{F}|$, whereas if $\vec{b}$ is a noisy encoding, it passes the check with probability at least $((N - n - 1)/N)^t$.

C AN IMPROVED PPRF FOR G_{SVOLE}

We describe in this section an improved (yet still relatively simple) puncturing strategy for constructing a t-puncturable PRF from the GGM PRF. This construction is somewhat folklore; it was explicitely presented in [17]. Unlike the simple construction presented in Section 4, however, this construction is not compatible with our distributed generation protocol. Still, it is useful in setting where computation is not an issue (hence a more costly distributed generation protocol can be used) but long-term storage is (hence it is important to reduce the size of the PCG keys), or in settings where a trusted dealer is available to distribute the PCG keys (like in the commodity-based model of Beaver [10]).

Intuitively, to obtain a t-puncturable PRF out of the GGM PRF, it suffices to define a key punctured at a subset S of leaves to be the smallest set of intermediate PRG values that allows to reconstruct all leaf values indexed by $[n] \setminus S$, and does not allow to reconstruct the leaf values indexed by S. We represent on Figure 13 a labelling algorithm which finds the indices of such a subset of the keys. The correctness of the algorithm follows easily by inspection; with a little more effort, one can also show that this algorithm is optimal (i.e., it produces the smallest possible punctured key satisfying the constraints). The worst-case scenario is easily seen to happen when all the punctured leaves are regularly spaces, with a distance of n/t between every two punctured leaves. This observation allows to upper bound the length of a key punctured at t points by $t\lambda \log(n/t)$, improving over the cost $t\lambda \log n$ of the naive approach.

Algorithm **Puncture-Label**

Input. A complete binary tree T with n leaves (indexed by $[n]$), and a size-t subset S of $[n]$. We denote by $s_1 < s_2 < \cdots < s_t$ the indices of the leaves in S.

Output. A labelling L_t of all nodes of T, such that all nodes of $[n] \setminus S$, and only them, belong to a subtree of T whose root belongs to L_t.

Procedure. The labelling proceeds in t steps. Given a leave x and a subtree T' of T which contains x, we denote by $\mathsf{Label}(x, T')$ the procedure which outputs all nodes of T' which have their parent node in P but are not in P themselves, where P denotes the path from the root of T' to x.

- In step 1, set $L_1 \leftarrow \mathsf{Label}(s_1, T)$.
- In step $i + 1$, let T_{i+1} denote the smallest subtree of T which contains s_{i+1} and whose root belongs to L_i (T_{i+1} exists by construction), and let r_{i+1} denote its root. Set $L_{i+1} \leftarrow (L_i \setminus \{r_{i+1}\}) \cup \{\mathsf{Label}(s_{i+1}, T_{i+1})\}$.

After all steps are completed, output L_t.

Figure 13: Labelling algorithm to compute the indices of a subset of keys in the GGM PRF construction which allows to reconstruct the output of the GGM PRF at all points except exactly t.

D NON-INTERACTIVE SECURE COMPUTATION WITH SILENT PREPROCESSING: PROTOCOL FLOW

To clarify the intended use and the features of our model for non-interactive secure computation protocols with silent preprocessing, we provide on Figure 14 a pictural representation of the protocol flow, illustrating the interdependencies between the algorithms, and we identify the main features of each of the algorithms (whether they require small communication, or only silent computation; whether they require cryptographic or non-cryptographic computation).

E SEMI-HONEST PCG AND NISC – MISSING PROOFS AND FIGURES

E.1 Missing Proofs

E.1.1 Proof of Theorem 5.1.

Correctness. We first show that the $s^i_{\alpha^*_i}$ values form a correct PRF key punctured at α. We need that for each i, $s^i_{\alpha^*_i}$ equals the GGM tree value that is sibling to the unique node on level i lying on the path to leaf α. This clearly holds for the first level, $i = 1$. On subsequent levels, R first computes the $2^i - 2$ values at level i that it can obtain from the previous values it knows, and then uses these to compute the final missing value $s^i_{\alpha^*_i}$. It does this by XORing (resp. summing over $\mathbb{F}_{p^r}$, for the last level) with the i-th OT output all-but-one of the odd-indexed, or even-indexed, values, depending

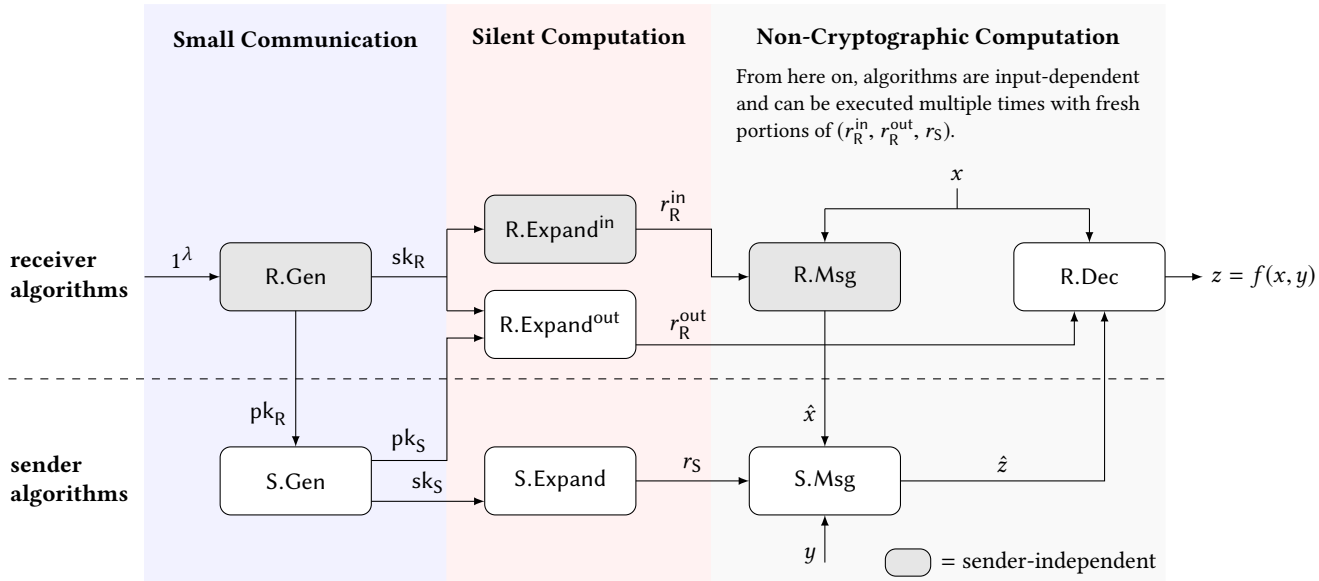

Figure 14: Pictural representation of the protocol flow for non-interactive secure computation with silent preprocessing. The receiver input is y, the sender input is x, and the target output is $z = f(x, y)$.

on the choice $\overline{\alpha_i}$. Since the sender's OT inputs contain the XOR (resp. sum over $\mathbb{F}_{p^r}$, for the last level) of every odd- or even-indexed value, the receiver ends up with the value of the sibling node to $s^i_{\alpha_i}$. To see correctness of the final correction value t, a similar reasoning as above shows that $t = \beta - \mathsf{PPRF.Eval}(k, \alpha)$, as required.

Security. We exhibit a simulator Sim that generates a view indistinguishable from an honest run of the protocol as long as a single party is corrupted.

Case 1: S is corrupted. The simulation of R is straightforward, since R does not send any message directly to S in the protocol; R only send inputs to $\mathcal{F}_{\mathsf{OT}}$.

Case 2: R is corrupted. Sim receives the input α and the target output $(\{s^i_{\alpha^*_i}\}_{i\in[\ell]}, t)$ of R. Sim defines $t^1 \cdots t^\ell$ and the values $(s^i_j)_{i,j}$ inductively, starting with $t^1 \leftarrow s^1_{\overline{\alpha_1}}$ and following the output procedure of R (see Figure 4). Eventually, Sim computes

$$c' = t - \left(t^\ell + \sum_{j=0, j\neq\alpha}^{2^{\ell-1}} s^\ell_{2j+\alpha_\ell} \right).$$

Then, Sim simulates the OT sender using input (t^i, d_i) as input if $\overline{\alpha_i} = 0$, and (d_i, t^i) as input otherwise, where d_i is an arbitrary dummy value; Sim also sends c' in parallel to the OTs. The indistinguishability of the simulation follows directly from the definition of $\mathcal{F}_{\mathsf{OT}}$ and by construction of c'.

E.1.2 Proof of Theorem 5.4.

Correctness. By the correctness of Π_{Gen} and G_{OT}, it holds that $v_i = w_{i,u_i}$ for $i = 1$ to n. Therefore, $m'_{i,s_i} - v_i = m_{i,s_i} + w_{i,t_i - s_i} - w_{i,u_i} = m_{i,s_i}$ since $t_i - s_i = u_i$.

Security. We exhibit a simulator Sim that generates a view indistinguishable from an honest run of the protocol as long as a single party is corrupted.

Case 1: S is corrupted. Sim simulates R by constructing $(\vec{e}, \vec{u})$ honestly, participating to Π_{Gen} as R does (note that this does not require any input of R). Sim simulates $\vec{t}$ by sending $\vec{t}' \xleftarrow{\$} \mathbb{F}_p^n$. Since Π_{Gen} securely emulates $\mathbb{F}_{\mathsf{Gen}}$, no information about $\vec{u}$ leaks to S during the execution of Π_{Gen}. By the security of G_{OT}, $\vec{u}$ is computationally indistinguishable from random from the viewpoint of S, hence so is $\vec{t} = \vec{u} + \vec{s}$; therefore, the simulation is indistinguishable from an honest run of the protocol.

Case 2: R is corrupted. Sim receives R's input $(s_i)_{i\leq n}$, R's random tape, and the corresponding target output $(m_{i,s_i})_i$ from the OT functionality. Sim simulates S by sampling k_1 and computing the $w_{i,j}$ honestly (this does not require the input of S). Sim computes the random noise vector $\vec{e}$ of R using R's random tape, from which he can compute R's output $\mathsf{k}_0 = (\vec{u}, \vec{v})$. For $i = 1$ to n, Sim computes m'_{i,s_i} as $m_{i,s_i} + v_i$, and picks $m'_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ for each $j \neq s_i$. Sim sends $(m'_{i,j})_{i,j}$ to R. By the security of Π_{Gen} and G_{OT}, the $m'_{i,j}$ for $j \neq s_i$ are indistinguishable from random from the viewpoint of R, hence the simulation is indistinguishable from an honest run of the protocol.

E.2 PCG for OT Correlation from Subfield-VOLE

We recall the construction of PCG for OT correlation from PCG for subfield-VOLE, introduced in [17], on Figure 15.

E.3 Semi-Honest NISC for OT with Silent Preprocessing

Our 2-round OT extension protocol gives rise to a non-interactive secure computation protocol for the oblivious transfer functionality, with silent preprocessing. We represent it on Figure 16.

Construction G_{OT}

PARAMETERS:

- Security parameter 1^λ, integers $n, q = p^r$.
- An $\mathbb{F}_p$-correlation-robust function $\mathsf{H} : \{0,1\}^\lambda \times \mathbb{F}_q \to \{0,1\}^\lambda$.
- The subfield-VOLE PCG $(G_{\mathsf{sVOLE}}.\mathsf{Gen}, G_{\mathsf{sVOLE}}.\mathsf{Expand})$

CORRELATION: Outputs $(R_0, R_1) = \left(\{(u_i, w_{i,u_i})\}_{i\in[n]}, \{w_{i,j}\}_{i\in[n], j\in[p]}\right)$, where $w_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ and $u_i \xleftarrow{\$} \{1,\ldots,p\}$, for $i \in [n], j \in [p]$.

GEN: On input 1^λ, output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Gen}(1^\lambda, n, p, q)$.

EXPAND: On input $(\sigma, \mathsf{k}_\sigma)$:

(1) If $\sigma = 0$: compute $(\vec{u}, \vec{v}') \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, where $\vec{u} \in \mathbb{F}_p^n, \vec{v}' \in \mathbb{F}_q^n$. Compute
$$v_i \leftarrow \mathsf{H}(i, v_i') \quad \text{for } i = 1,\ldots,n$$
and output (u_i, v_i).

(2) If $\sigma = 1$: compute $(x, \vec{w}') \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, where $x \in \mathbb{F}_q, \vec{w}' \in \mathbb{F}_q^n$. Compute
$$w_{i,j} \leftarrow \mathsf{H}(i, w_i' - j \cdot x) \quad \text{for } i = 1,\ldots,n, \forall j \in \mathbb{F}_p$$
and output $\{w_{i,j}\}_{i,j}$.

Figure 15: PCG for n sets of 1-out-of-p random OT

F DETAILS FOR MALICIOUSLY SECURE CONSTRUCTIONS

F.1 Definition and GGM Instantiation of Punctured PRF with Malicious Keys

Formally, we consider a key verification property of the underlying PPRF given in Definition F.1 below. For our application, the malicious keyspace $\mathcal{K}$ will correspond to malicious choices of the sender's ℓ OT message pairs. For a given malicious key K and subset of the PRF domain $I \subset [N]$, $\mathsf{Ver}(K, I)$ evaluates to 1 if the full-domain evaluation vector $\vec{s} = (s_0, \ldots, s_{N-1})$ of K (as defined by Eval^*) is "well formed" for I: namely, for any possible choice of the receiver's input $\alpha \in I$, then the sender's string $\vec{s}$ agrees with the corresponding receiver full-domain evaluation string, defined by the received key k^* derived from puncturing K (via $\mathsf{Puncture}^*$) at α.

Definition F.1 (Verification of malicious PPRF keys). Let (PPRF.Gen, PPRF.Puncture, PPRF.Eval) be a PPRF with keyspace $\{0,1\}^\lambda$, domain $\mathcal{X}$ and range $\mathcal{Y}$. We say that PPRF allows *verification of malicious keys* for a set $\mathcal{K}$, the malicious keyspace, if there exist efficient algorithms ($\mathsf{Ver}, \mathsf{Puncture}^*, \mathsf{Eval}^*$), such that;

- Ver takes as input a malicious key $K \in \mathcal{K}$ and a set $I \subseteq \mathcal{X}$ and outputs 0/1.
- $\mathsf{Puncture}^*$ takes as input a malicious key K and an index $\alpha \in \mathcal{X}$ and outputs a key $\mathsf{k}^*_{\mathsf{pprf}}$ punctured at α.
- Eval^* takes as input a malicious key K^*, a set $I \subseteq \mathcal{X}$ and an index in $x \in \mathcal{X}$, and outputs a value in $\mathcal{Y}$ or $\perp$.

Protocol $\mathsf{NISC}_{\mathsf{OT}}$

PARAMETERS: $1^\lambda, n, N = 2^\ell, t, p, r \in \mathbb{N}$. $H \in \mathbb{F}_p^{N\times n}$. PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{p^r}$. $\mathcal{D}_{t,N}$ is a weight-t error distribution over $\mathbb{F}_p^N$.

$\mathsf{par} \leftarrow (1^\lambda, n, N, t, p, r, H)$.

INPUTS:

- R inputs n field elements $(s_i)_{i\le n} \in \mathbb{F}_p^n$.
- S inputs n length-p vectors $(\vec{m}_i)_{i\le n}$ where each $\vec{m}_i$ is over $(\{0,1\}^\lambda)^p$.

PROTOCOL:

- $\mathsf{R.Gen}(\mathsf{par})$: pick $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Compute the first round of Π_{Gen} on input $\vec{e}$. Set sk_R to be the secret state of R after computing the first round of Π_{Gen}, and pk_R be the message computed by R in Π_{Gen}.
- $\mathsf{R.Expand}^{\mathsf{in}}(\mathsf{sk}_\mathsf{R})$: compute the first part $\vec{u}$ of $G_{\mathsf{OT}}.\mathsf{Expand}(0, \mathsf{k}_0)$ (note that $\vec{u}$ is computed as $\vec{e} \cdot H$ where $\vec{e}$ depends solely on $\vec{e}$). Output $r^{\mathsf{in}}_\mathsf{R} \leftarrow \vec{u}$.
- $\mathsf{S.Gen}(\mathsf{pk}_\mathsf{R})$: sample $x \xleftarrow{\$} \mathbb{F}_{p^r}$ and $\mathsf{k}_{\mathsf{pprf}} \xleftarrow{\$} \mathsf{PPRF.Gen}(1^\lambda)$. Set $\mathsf{sk}_\mathsf{S} \leftarrow (\mathsf{k}_{\mathsf{pprf}}, x)$. Define pk_S to be the second round message of S in Π_{Gen} on input $(x, \mathsf{k}_{\mathsf{pprf}})$ after receiving the message pk_R from R.
- $\mathsf{R.Expand}^{\mathsf{out}}(\mathsf{sk}_\mathsf{R}, \mathsf{pk}_\mathsf{S})$: compute the output $r^{\mathsf{out}}_\mathsf{R}$ obtained by R with state sk_R upon receiving the message pk_S from S in Π_{Gen}.
- $\mathsf{S.Expand}(\mathsf{sk}_\mathsf{S})$: compute $r_\mathsf{S} = \{w_{i,j}\}_{i\le n, j\le p} \leftarrow G_{\mathsf{OT}}.\mathsf{Expand}(1, \mathsf{sk}_\mathsf{S})$.
- $\mathsf{R.Msg}(r^{\mathsf{in}}_\mathsf{R}, \vec{s})$: output $\hat{s} \leftarrow r^{\mathsf{in}}_\mathsf{R} + \vec{s}$.
- $\mathsf{S.Msg}(r_\mathsf{S}, \hat{s}, (\vec{m}_i)_i)$: parse r_S as $\{w_{i,j}\}_{i\le n, j\le p}$. Compute $m'_{i,j} \leftarrow m_{i,j} + w_{i,\hat{s}_i - j}$ for $i = 1$ to n and $j = 1$ to p and output $\hat{z} = (m'_{i,j})_{i,j}$.
- $\mathsf{R.Dec}(r^{\mathsf{out}}_\mathsf{R}, \vec{s}, \hat{z})$: parse $\hat{z}$ as $(m'_{i,j})_{i,j}$. Compute $(\vec{u}, \vec{v}) \leftarrow G_{\mathsf{OT}}.\mathsf{Expand}(0, r^{\mathsf{out}}_\mathsf{R})$ and output $z = (m'_{i,s_i} - v_i)_{i\le n}$.

Figure 16: Non-interactive secure computation with silent preprocessing for oblivious transfer

Futher, we require for all $I \subseteq \mathcal{X}$ and $K^* \in \mathcal{K}$:

Consistency check. If $\mathsf{Ver}(K^*, I) = 1$ then for all $\alpha \in I, x \in \mathcal{X} \setminus \{\alpha\}$: $\mathsf{PPRF.PuncEval}(\mathsf{k}^*, x) = \mathsf{Eval}^*(K^*, I, \alpha)$, where $\mathsf{k}^* \leftarrow \mathsf{Puncture}^*(K^*, \alpha)$.

If this holds then we say that K^* is *consistent* with the set I.

GGM instantiation. We use the GGM puncturable PRF with domain $[2N]$ and range $(\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$. The underlying PRG of the first ℓ levels we denote by $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. The PRG of the last level we denote by $G' : \{0,1\}^\lambda \to (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$. In the following by $\mathsf{PPRF}_{\mathsf{GGM}} = (\mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{Puncture}, \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{Eval}, \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{PuncEval})$ we denote the standard GGM

PPRF algorithms. By $\mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{FullEval}(\mathsf{k}^*_{\mathsf{pprf}})$ we denote the algorithm that on input of a $\mathsf{k}^*_{\mathsf{pprf}}$ punctured at $\alpha \in [2N]$ evaluates the PRF on all values except α and returns the output values $\{s_j\}_{j\in[2N]\setminus\{\alpha\}}$.

In the following we explain how the GGM construction allows verification of malicious PPRF keys. We set the malicious key space to $\mathcal{K} = \{0,1\}^{2\lambda\ell} \times ((\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda)$. We restrict the puncturing algorithm $\mathsf{Puncture}^*$ to even inputs $\alpha\|0 \in [2N]$, as these will later correspond to the actual output values. Note that the key $K_0^{\ell+1} \in \mathbb{F}_{p^r}^2$ takes a special role. Namely, with this key we capture a programmibility at the punctured point when it would be undefined otherwise: If $I = \{\alpha\}$ consists only of a single point, then the output value at this point α is set to be $K_0^{\ell+1}$ by Eval^* (if $|I| > 1$ this value is ignored).

Ver: On input $K = \{(K_0^i, K_1^i)_{i=1}^{\ell+1}\}$ and $I \subseteq [N]$, compute $\mathsf{k}^*_\alpha \leftarrow \mathsf{Puncture}^*(K,\alpha)$ for all $\alpha \in I$. Return 1, if and only if for all $\alpha, \alpha' \in I, x \in [2N]\setminus\{\alpha\|0, \alpha'\|0\}$ it holds:

$$\mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{PuncEval}(\mathsf{k}^*_\alpha, x) = \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{PuncEval}(\mathsf{k}^*_{\alpha'}, x).$$

$\mathsf{Puncture}^*$: On input $\{(K_0^i, K_1^i)_{i=1}^{\ell+1}\}$ and $\alpha \in [N]$, we set $\alpha_{\ell+1} = 0$ and proceed as follows: Let $\alpha_i^* = \alpha_1 \cdots \alpha_{i-1}\overline{\alpha_i}$ for all $i \in \{1,\dots,\ell+1\}$.

(1) Define $s^1_{\alpha_1^*} = K^1_{\overline{\alpha_1}}$.

(2) For $i \in \{2,\dots,\ell\}$:

(a) Compute $(s^i_{2j}, s^i_{2j+1}) = G(s_j^{i-1})$, for $j \in [0,\dots,2^{i-1})$, $j \neq \alpha_1 \cdots \alpha_{i-1}$ and $(s^{\ell+1}_{2j}, s^{\ell+1}_{2j+1}) = G'(s^\ell_j)$ for $j \in [N], j \neq \alpha$.

(b) For $i \in \{1,\dots,\ell+1\}$ compute:

$$s^i_{\alpha_i^\star} = K^i_{\overline{\alpha_i}} \oplus \bigoplus_{\substack{j\in[0,2^{i-1}),\\ j\neq\alpha_i^*}} s^i_{2j+\overline{\alpha_i}} \in \{0,1\}^\lambda.$$

(3) Output the punctured key $\mathsf{k}^*_{\mathsf{pprf}} = \{s^i_{\alpha_i^\star}\}_{i=1}^{\ell+1}$.

For better readablity, we overload notation and denote by $\mathsf{Puncture}^*$ also the procedure taking only the required keys $\{K^i_{\overline{\alpha_i}}\}_{i=1}^{\ell+1}$ as input.

Eval^*: On input $K = \{(K_0^i, K_1^i)_{i=1}^{\ell+1}\}$,$I$ and $x \in [2N]$ proceed as follows:

(1) If $\mathsf{Ver}(K, I) = 0$, return $\perp$.

(2) If $\hat{I} = \{\alpha\|0 : \alpha \in I\}$ consists of the single point x return $K_0^{\ell+1} \in \mathbb{F}_{p^r}^2$

(3) Else, compute $\mathsf{k}^* \leftarrow \mathsf{Puncture}^*(K,\alpha)$ for some $\alpha\|0 \in \hat{I}\setminus\{x\}$ and return $(\omega, w) \leftarrow \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{PuncEval}(\mathsf{k}^*, x)$.

Again, we overload notation and for $\alpha \in [N]$ denote by $\mathsf{Eval}^*(K, I, \alpha)$ also the algorithm that calls $\mathsf{Eval}^*(K, I, x)$ for $x = \alpha\|0 \in [2N]$.

It is left to show that the algorithms indeed allow verification of malicious keys. Note that if $\mathsf{Ver}(K^*, I) = 1$, then for all $\alpha \in I, x \in [2N]\setminus\{\alpha\|0\}$ we have: $\mathsf{Eval}^*(K^*, I, x) = \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{PuncEval}(\mathsf{k}^*, x)$, where $\mathsf{k}^* \leftarrow \mathsf{Puncture}^*(K^*, \alpha)$. By step (3) in the verification procedure, we have that this value is independent of the choice of α. This yields the required.

F.2 Malicious Setup for Single-Point PPRF

Definition F.2. Right-half injectivity We say a function $f = (f_0, f_1)\colon \{0,1\}^\lambda \to \mathcal{Y}\times\{0,1\}^\lambda, x \mapsto (f_0(x), f_1(x))$ is *right-half injective*, if its restriction to the right-half output $f_1\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ is injective.

Formally, we define PPRF PPRF_1 with domain $[N]$ and range $(\mathbb{F}_{p^r})^2$ as follows: Let $\mathsf{PPRF}_1 = (\mathsf{PPRF}_1.\mathsf{Puncture}, \mathsf{PPRF}_1.\mathsf{Eval})$, such that:

$\mathsf{PPRF}_1.\mathsf{Puncture}$**:** On input $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$ and $\alpha \in [N]$, return $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{Puncture}(\mathsf{k}_{\mathsf{pprf}}, \alpha\|0)$.

$\mathsf{PPRF}_1.\mathsf{Eval}$**:** On input $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$ and $\alpha \in [N]$, return $(\omega, w) \leftarrow \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha\|0)$.

We give the protocol for distributed setup of PPRF_1 with security against malicious adversaries in Figure 17.

Note that steps (1) to (6) correspond to the protocol in the semi-honest case with an additional level of PRG evaluation, where for the last level always the sum of the right leaves are given to the receiver. This will allow the receiver to check the hash value computed by the sender in step (7).

Theorem 6.1. *Assuming a black-box access to a PRG $G : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$, a right-half injective PRG $G' : \{0,1\}^\lambda \mapsto (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$, and a collision resistant hash function $h\colon \{0,1\}^{\lambda N} \to \{0,1\}^\lambda$, there exists a 2-party protocol implementing $\mathcal{F}_{\mathsf{m\text{-}PPRF}}$ (see Fig. 18) for the puncturable PRF PPRF_1, with malicious security in the parallel OT-hybrid model, and the following efficiency features. The interaction consists of ℓ parallel calls to $\mathcal{F}_{\mathsf{OT}}$, and uses additional communication of $r \log p + \lambda$. The computational complexity is dominated by $O(2^\ell)$ calls each to G and G'.*

Proof. When neither party is corrupted, the receiver R will indeed compute the correct punctured key $\mathsf{k}^*_{\mathsf{pprf}}$ and output value w in a protocol execution (the proof is similar to the semi-honest case and will be shown in more detail in the paragraph for security against malicious receiver.) Further, as the tree is punctured at an even value $\alpha\|0$, both parties hold the same values $\gamma_0,\dots,\gamma_{N-1}$ for all odd leaves. Therefore, $\Gamma = \Gamma'$ in step (6d) of the protocol execution.

Security against malicious receiver. On input α, the simulator forwards α to the functionality. On output $\mathsf{k}^*_{\mathsf{pprf}} = \{k_i\}_{i=1}^{\ell+1}$ and w, the simulator proceeds as follows:

- For $i \in \{1,\dots,\ell\}$ define $s^i_{\alpha_1,\dots,\alpha_{i-1}\overline{\alpha_i}} = k_i$ and $s^{\ell+1}_{\alpha\|1} = k_{\ell+1}$. Set $K^1_{\overline{\alpha_1}} = s^1_{\overline{\alpha_1}}$.
- For $i \in \{2,\dots,\ell\}$: Compute $(s^i_{2j}, s^i_{2j+1}) = G(s_j^{i-1})$, for $j \in [0,\dots,2^{i-1}), j \neq \alpha_1,\dots,\alpha_{i-1}$.
- Compute $(s^{\ell+1}_{2j}, s^{\ell+1}_{2j+1}) = G'(s^\ell_j)$, for $j \in [0,\dots,2^\ell), j \neq \alpha\|0$.
- For $i \in \{1\dots,\ell+1\}$: Compute

$$K^i_{\overline{\alpha_i}} = \bigoplus_{j\in[0,2^{i-1})} s^i_{2j+\overline{\alpha_i}},$$

where we set $\alpha_{\ell+1} = 0$.

Protocol $\Pi_{\text{m-PPRF}}$:

PARAMETERS: $1^\lambda, \ell, N = 2^\ell, p, r \in \mathbb{N}$. $\mathsf{PPRF}_{\mathsf{GGM}}$ is the GGM puncturable PRF with domain $\{0,1\}^{\ell+1} = [2N]$, key space $\{0,1\}^\lambda$, and range $(\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$, constructed from a length-doubling PRG $G : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$, and a second PRG $G' : \{0,1\}^\lambda \mapsto (\mathbb{F}_{p^r})^2 \times \{0,1\}^\lambda$ used to compute the PRF outputs on the last level of the tree.

INPUTS:

- R inputs $\alpha \in \{0,1\}^\ell$.
- S inputs $\beta \in (\mathbb{F}_{p^r})^2$ and a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.

PROTOCOL:

(1) S samples a random seed $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.
(2) S computes the 2^i partial evaluations at level i of the GGM tree:
  (a) S sets $s_0^0 = \mathsf{k}_{\mathsf{pprf}}$.
  (b) For $i \in \{1,\dots,\ell\}, j \in [0,\dots,2^{i-1})$: S computes $(s_{2j}^i, s_{2j+1}^i) = G(s_j^{i-1})$.
  (c) For $j \in \{0,1\}^\ell$: S computes $(s_{2j}^{\ell+1}, s_{2j+1}^{\ell+1}) = G'(s_j^\ell) \in (\mathbb{F}_{p^r})^2 \times \{0,1\}^\ell$.
(3) S computes the "left" and "right" halves for $i \in \{1,\dots,\ell\}$:
$$K_0^i = \bigoplus_{j\in[0,2^{i-1})} s_{2j}^i, \quad K_1^i = \bigoplus_{j\in[0,2^{i-1})} s_{2j+1}^i$$
(4) S computes the "right" half for $i = \ell+1$:
$$K_1^{\ell+1} = \bigoplus_{j\in\{0,1\}^\ell} s_{2j+1}^{\ell+1}$$
(5) For $i = 1,\dots,\ell = \log N$ (in parallel) the parties run OT where in the i-th OT:
  (a) R inputs the choice bit $\overline{\alpha_i}$.
  (b) S inputs the pair (K_0^i, K_1^i).
(6) S sends to R the key $K_1^{\ell+1}$ and the correction value
$$c = \beta - \sum_{j\in[N]} s_{2j}^{\ell+1}.$$
(7) For the consistency check, S sets $\gamma_j = s_{2j+1}^{\ell+1}$ for all $j \in [N]$ and sends to R the value $\Gamma = h(\gamma_0,\dots,\gamma_{N-1})$.
(8) Let $\{K^i\}_{i=1}^{\ell+1}$ denote the OT outputs received by R together with the key of the $(\ell+1)$-st level. Then, R proceeds as follows.
  (a) $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{Puncture}^*(\{K^i\}_{i=1}^{\ell+1}, \alpha)$.
  (b) $\{s_j\}_{j\neq\alpha\|0} \leftarrow \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{FullEval}(\mathsf{k}^*_{\mathsf{pprf}}, \alpha\|0)$.
  (c) R receives c, and computes
$$w = c - \sum_{j\in[N]\setminus\{\alpha\}} s_{2j}$$
  (d) To verify consistency, R sets $\gamma_j = s_{2j+1}$ for all $j \in [N]$, and computes $\Gamma' = h(\gamma_0,\dots,\gamma_{N-1})$.
(9) If $\Gamma = \Gamma'$, R outputs the punctured key $\mathsf{k}^*_{\mathsf{pprf}}$, and the final correction value w. Otherwise, R aborts.

Figure 17: Protocol for distributed setup of single-point PPRF with consistency check

Functionality $\mathcal{F}_{\text{m-PPRF}}$:

PARAMETERS: $1^\lambda, N = 2^\ell, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N] = \{0,1\}^\ell$, key space $\{0,1\}^\lambda$, and range $(\mathbb{F}_{p^r})^2$, supporting verification of malicious keys.

INPUTS:

- R inputs $\alpha \in \{0,1\}^\ell$.
- S inputs $\beta \in (\mathbb{F}_{p^r})^2$ and a PPRF key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$.

FUNCTIONALITY:

- If S is honest:
  (1) Compute $\mathsf{k}^*_{\mathsf{pprf}} = \mathsf{PPRF.Puncture}(\mathsf{k}_{\mathsf{pprf}}, \alpha)$.
  (2) Send $\mathsf{k}^*_{\mathsf{pprf}}$ and $w = -\mathsf{PPRF.Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha) + \beta$ to R, and $\mathsf{k}_{\mathsf{pprf}}$ to S.
- If S is corrupted:
  (1) Wait for $\mathcal{A}$ to send a guess $I \subseteq [N]$ and a key $K^* \in \mathcal{K}$.
  (2) Check that $\alpha \in I$ and that $\mathsf{Ver}(K^*, I) = 1$. If either check fails, send *abort* to R and wait for a response from R. When R responds with *abort*, forward this to S and halt.
  (3) Compute $\mathsf{k}^*_{\mathsf{pprf}} = \mathsf{Puncture}^*(K^*, \alpha)$ and
$$w = -\mathsf{Eval}^*(K^*, I, \alpha) + \beta.$$
  (4) Send $\mathsf{k}^*_{\mathsf{pprf}}$ and w to R, and *success* to S.

Figure 18: Functionality for malicious distributed setup of single-point PPRF

- Compute
$$c = w - \sum_{j\in[N]\setminus\{\alpha\}} s_{2j}^{\ell+1}.$$
- Set $\gamma_j = s_{2j+1}^{\ell+1}$ for $j \in [N]$, and compute $\Gamma = h(\gamma_0,\dots,\gamma_{N-1})$.

Finally, the simulator forwards $\{K^i_{\overline{\alpha_i}}\}_{i=1}^{\ell}$ (where the i-th key corresponds to the i-th OT message), $K_1^{\ell+1}$, c and Γ to the receiver.

Note that the values $\{K^i_{\overline{\alpha_i}}\}_{i=1}^{\ell+1}$ and Γ correspond to the values computed by the sender in a real protocol execution. We have to show that the key $\mathsf{k}'^*_{\mathsf{pprf}} = \{s'_{\alpha_i^*}\}_{i=1}^{\ell+1} \leftarrow \mathsf{Puncture}^*(\{K^i\}_{i=1}^{\ell+1}, \alpha)$ computed by the receiver in a protocol execution corresponds to the key $\mathsf{k}^*_{\mathsf{pprf}} = \{k_i\}_{i=1}^{\ell+1}$ returned by the functionality. We have:

- $s'^1_{\alpha_1^\star} = K^1_{\overline{\alpha_1}} = k_1$.
- Assume that we have $s'^{i-1}_j = s^{i-1}_j$ for all $j \in [0,\dots,2^{i-1})$, $j \neq \alpha_1,\dots,\alpha_{i-1}$ for some i. Then we have the same for level i, as this is true for all values off the path, and further because of
$$s'^i_{\alpha_i^*} = K^i_{\overline{\alpha_i}} \oplus \bigoplus_{\substack{j\in[0,2^{i-1}),\\ j\neq\alpha_i^*}} s^i_{2j+\overline{\alpha_i}} = s^i_{\alpha_i^*}.$$
  As $s^i_{\alpha_i^*} = k_i$ for all $i \in \{1,\dots,\ell_1\}$, the keys agree as required.

Further, it holds

$$c = w - \sum_{j\in[N]\setminus\{\alpha\}} s_{2j}^{\ell+1} = -s_{\alpha\|0}^{\ell+1} + \beta - \sum_{j\in[N]\setminus\{\alpha\}} s_{2j}^{\ell+1}$$
$$= \beta - \sum_{j\in[N]} s_{2j}^{\ell+1},$$

where $s_{\alpha\|0}^{\ell+1} \leftarrow \mathsf{PPRF}_{\mathsf{GGM}}.\mathsf{Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha\|0)$ (for the key $\mathsf{k}_{\mathsf{pprf}} \in \{0,1\}^\lambda$ corresponding to $\mathsf{k}^*_{\mathsf{pprf}}$).

Security against malicious sender. On input of the OT messages $K = \{(K_0^i, K_1^i)_{i=1}^\ell\}$, key $K_1^{\ell+1}$, correction value c and a hash value Γ, the simulator proceeds as follows: First, the simulator computes Γ_α for each $\alpha \in [N]$ as follows:

(1) Compute $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{Puncture}^*(\{K^i_{\overline{\alpha_i}}\}_{i=1}^{\ell+1}, \alpha)$.
(2) Compute $\{s_j(\alpha)\}_{j\in[2N]\setminus\{\alpha\|0\}} \leftarrow \mathsf{PPRF.FullEval}(\mathsf{k}^*_{\mathsf{pprf}}, \alpha\|0)$.
(3) Compute
$$w = c - \sum_{j\in[N]\setminus\{\alpha\}} s_{2j}(\alpha)$$
(4) Set $\gamma_j(\alpha) = s_{2j+1}(\alpha)$ for $j \in [N]$, and compute $\Gamma'_\alpha = h(\gamma_0(\alpha), \ldots, \gamma_{N-1}(\alpha))$.

Note that this is exactly how an honest receiver, on input α, would proceed. If the sender behaved honestly, we should have $\Gamma_1 = \Gamma_2 \cdots = \Gamma_N = \Gamma$.

Let $I \subset [n]$ be the set of α's consistent with Γ, that is

$$I = \{\alpha \in [n] \mid \Gamma_\alpha = \Gamma\}$$

If $I = \emptyset$ then abort. We extract the sender's input β as follows:

- Pick an $\alpha \in I$.
- For all $j \in [2N]\setminus\alpha\|0$, define $s_j = s_j(\alpha)$.
- If $|I| = 1$, set $s_{\alpha\|0} = 0$.
- Otherwise, pick $\alpha' \in I$ (with $\alpha' \neq \alpha$), define $s_{\alpha\|0} = s_{\alpha\|0}(\alpha')$.
- Set
$$\beta = -s_{\alpha\|0} + w.$$
- Input $\beta, K^* = \{K, (0, K_1^{\ell+1})\}, I$ to the functionality.

We have $\mathsf{Ver}(K, I) = 1$ because of the following.

Claim F.3. *Except with negligible probability, all choices of $\alpha, \alpha' \in I$ in the above procedure lead to the same vector $\vec{s} = (s_0, \ldots, s_{N-1})$.*

Proof. The case $|I| = 1$ is trivial. For $|I| > 1$, it suffices to show that for all $\alpha, \alpha' \in I$ and $j \in [2N] \setminus \{\alpha\|0, \alpha\|0'\}$, $s_j(\alpha) = s_j(\alpha')$. Suppose for a contradiction that this does not hold, so there exist $j \in [2N], \alpha, \alpha' \in [N]$ such that $s_j(\alpha) \neq s_j(\alpha')$. From the fact that $\Gamma_\alpha = \Gamma_{\alpha'}$ and the collision-resistance of h, we have $\gamma_i(\alpha) = \gamma_i(\alpha')$ for all $i \in [N]$, except with negligible probability. Recall that each $(s_{2i}(\alpha), s_{2i+1}(\alpha)), (s_{2i}(\alpha'), s_{2i+1}(\alpha'))$, where $i \notin \{\alpha, \alpha'\}$, we have $(s_{2i}(\alpha), s_{2i+1}(\alpha)) = G'(\rho)$ and $(s_{2i}(\alpha'), s_{2i+1}(\alpha')) = G'(\rho')$, for some ρ, ρ'. From the right-half injectivity of G', we have that, if $s_{2i+1}(\alpha) = \gamma_i = \gamma'_i = s_{2i+1}(\alpha')$ then it must hold that $\rho = \rho'$. Hence, we must have $s_j(\alpha) = s_j(\alpha')$ for all $j \in [2N] \setminus \{\alpha\|0, \alpha\|0'\}$, which completes the claim.

□

Functionality $\mathcal{F}_{\text{g-rev-VOLE}}$

Parameters: $t, p, r \in \mathbb{N}$.

Input: The sender S inputs a pair $((\vec{\beta}, \chi), (\vec{b}, x)) \in (\mathbb{F}_{p^r}^t \times \mathbb{F}_{p^r})^2$. The receiver R inputs a vector $\vec{y} \in \mathbb{F}_p^t$.

Functionality: Compute $\vec{\gamma} \leftarrow \vec{y}\chi - \vec{\beta}$ and $\vec{c} \leftarrow \vec{y}x - \vec{b}$ and output $(\vec{\gamma}, \vec{c})$ to R.

Figure 19: Generalized Reverse Vector-OLE Functionality over a Field $\mathbb{F}_p$

Note that the punctured key returned by the functionality equals the key computed in the real protocol execution. Next, we show that the correction value w corresponds to the one computed in the real protocol execution. For $|I| = 1$ this follows, as $\mathsf{Eval}^*(K^*, I, \alpha) = 0$. For $|I| > 1$, this follows as for all $\alpha \in I, x \in I\setminus\{\alpha\}$, $\mathsf{k}^*_{\mathsf{pprf}} \leftarrow \mathsf{Puncture}^*(K, x)$, it holds $\mathsf{PPRF.Eval}(\mathsf{k}^*_{\mathsf{pprf}}, \alpha\|0) = s_{\alpha\|0}$ and thus $-\mathsf{PPRF.Eval}(\mathsf{k}_{\mathsf{pprf}}, \alpha\|0) + \beta = -s_{\alpha\|0} + \beta = w$ as required.

In the real execution the receiver aborts, if $\Gamma' \neq \Gamma$. By previous considerations this is equivalent to $\alpha \in I$. It follows that the functionality aborts if and only if the real protocol execution would have aborted.

□

F.3 Malicious Setup of t PPRFs with Consistent Offset

Theorem 6.2. *There exists a 4-message 2-party protocol $\Pi_{\text{m-Gen}}$ (see Fig. 21 in Section F.2) which securely implements the functionality $\mathcal{F}_{\text{m-Gen}}(1^\lambda, N, p, r)$ (see Fig. 20 in Section F.2) for the puncturable PRF PPRF in the $\mathcal{F}_{\text{g-rev-VOLE}}$-, parallel $\mathcal{F}_{\text{m-PPRF}}$-hybrid model, with malicious security, using t parallel calls to $\mathcal{F}_{\text{m-PPRF}}$, and only one call to $\mathcal{F}_{\text{g-rev-VOLE}}$, and further communication of $(N + t + 2)r \log p$ bits. Furthermore, when $p = 2$, the functionality can be implemented in the parallel $\mathcal{F}_{\text{m-PPRF}}$-hybrid model, using no call to $\mathcal{F}_{\text{g-rev-VOLE}}$.*

Using an additional pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \mathbb{F}_{p^r}^{N+1}$, the communication can be reduced to just $(t + 1)r \log p + \lambda$ bits, by sampling τ, τ_i using a random seed for PRG. We did not include this in the protocol, to simplify its description and proof of security.

Proof. If both parties are honest, after execution of the protocol, R and S hold values $\vec{v}_R^j = (v_{R,0}^j, v_{R,2}^j, \ldots, v_{R,2N-2}^j)$, $\vec{v}_R'^j = (v_{R,1}^j, v_{R,3}^j, \ldots, v_{R,2N-1}^j)$ and $\vec{v}_S^j = (v_{S,0}^j, v_{S,2}^j, \ldots, v_{S,2N-2}^j)$, $\vec{v}_S'^j = (v_{S,1}^j, v_{S,3}^j, \ldots, v_{S,2N-1}^j)$, such that

$$\vec{v}_R^j + \vec{v}_S^j = (\beta_j + \gamma_j)\vec{e}_{\alpha_j} = \chi \cdot y_j \cdot \vec{e}_{\alpha_j} \text{ and } \vec{v}_R'^j + \vec{v}_S'^j = x \cdot y_j \cdot \vec{e}_{\alpha_j},$$

and thus $(\vec{v}_R^j + \tau \cdot \vec{v}_R'^j) + (\vec{v}_S^j + \tau \cdot \vec{v}_S'^j) = X \cdot y_j \cdot \vec{e}_{\alpha_j}$, where $X = \chi + \tau \cdot x$. Computing the scalar product with $(\tau_0, \ldots, \tau_{N-1})$ on both sides of the equation yields the required.

Security against malicious receiver: Receive from $\mathcal{A}$ input $y = (y_1, \ldots, y_t) \in \mathbb{F}_p$ to $\mathcal{F}_{\text{rev-VOLE}}$ and (not necessarily distinct) inputs $\alpha_1, \ldots, \alpha_t \in [N]$ to $\mathcal{F}_{\text{m-Gen}}$. For $j \in [t]$ we define $\vec{e}_{\alpha_j}$ to be the α_j-th unit vector, and $\vec{u} = \sum_{i=1}^t y_j \vec{e}_j$. The simulator forwards $\alpha_1, \ldots, \alpha_t$

Functionality $\mathcal{F}_{\text{m-Gen}}$:

PARAMETERS: $1^\lambda, N = 2^\ell, t, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N]$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}_{p^r}$, supporting verification of malicious keys.

INPUTS:

- R inputs indices $\alpha_1, \ldots, \alpha_t \in [N]$ and weights $y_1, \ldots, y_t \in \mathbb{F}_p^*$. We define $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\}$.
- S inputs $\mathsf{k}_1 = (\{\mathsf{k}_j\}_{j\in[t]}, x)$, where $x \in \mathbb{F}_{p^r}$ and $\mathsf{k}_j \in \{0,1\}^\lambda$.

FUNCTIONALITY:

- If S is honest:
  1. Compute $\mathsf{k}_j^* \leftarrow \mathsf{PPRF.Puncture}(\mathsf{k}_j, \alpha_j)$, for $j \in \{1, \ldots, t\}$.
  2. Let $z_j \leftarrow x \cdot y_j - \mathsf{PPRF.Eval}(\mathsf{k}_j, \alpha_j)$ for $j \in \{1, \ldots, t\}$.
  3. Let $\mathsf{k}_0 \leftarrow (\{\mathsf{k}_j^*, z_j\}_{j\in[t]}, \mathcal{S}, \vec{y})$.
  4. Output k_0 to R.
- If S is corrupted:
  1. Receive from $\mathcal{A}$ t subsets $I_1, \ldots, I_t \subseteq [N]$ and a set of keys $K_1^*, \ldots, K_t^* \in \mathcal{K}$.
  2. For each $j \in \{1, \ldots, t\}$ check that $\alpha_j \in I_j$ and that $\mathsf{Ver}(K_j^*, I_j) = 1$. If any check fails, abort.
  3. Compute $k_j^* = \mathsf{PPRF.Puncture}^*(K_j^*, \alpha_j)$ for each $j \in \{1, \ldots, t\}$.
  4. Let $z_j \leftarrow x \cdot y_j - \mathsf{PPRF.Eval}^*(K_j^*, I_j, \alpha_j)$ for $j \in \{1, \ldots, t\}$.
  5. Output $\mathsf{k}_0 \leftarrow (\mathsf{k}_{\mathsf{pprf}}^*, \mathcal{S}, \vec{y}, \{z_j\}_{j\in[t]})$ to R and *success* to S.

Figure 20: Functionality for malicious distributed setup of t puncturable PRFs

and $y_1, \ldots, y_t$ to $\mathcal{F}_{\text{m-Gen}}$, and for each $j \in \{1, \ldots, t\}$ forwards k_j^* and z_j to $\mathcal{A}$ for each $j \in \{1, \ldots, t\}$. On inputs $\tau, \tau_0, \ldots, \tau_{N-1}$ of $\mathcal{A}$, the simulator draws $X \xleftarrow{\$} \mathbb{F}_{p^r}$, computes for $i \in [N], j \in \{1, \ldots, t\}$:

$$(v_{R,2i}^j, v_{R,2i+1}^j) \leftarrow \mathsf{PPRF.Eval}'(\mathsf{k}_j^*, i)$$

and sends X and $V_{S,j} = X \cdot \tau_{\alpha_j} \cdot y_j - \sum_{i=0}^{N-1} \tau_i \cdot (v_{R,2i}^j + \tau \cdot v_{R,2i+1}^j)$ to $\mathcal{A}$. It is left to show that this is indistinguishable from a real protocol execution. Let x be the input of the sender to the functionality. We set $\chi = X - \tau \cdot x$. For $j \in \{1, \ldots, t\}$ we have $\gamma_j + \beta_j = \chi \cdot y_j$, $c_j + b_j = x \cdot y_j$, and $(\omega_j, w_j) = -\mathsf{PPRF.Eval}(\mathsf{k}_j, \alpha_j) + (\beta_j, b_j)$ (since S is honest). Thus, we have

$$(\omega_j, w_j) + (\gamma_j, c_j) = -\mathsf{PPRF.Eval}(\mathsf{k}_j, \alpha_j) + (x \cdot y_j, \chi \cdot y_j).$$

For $\mathsf{PPRF.Eval}'$ (as defined in Figure 17) for all $i \in [N]$ this yields

$$\mathsf{PPRF.Eval}'(\mathsf{k}_j^*, i) = -\mathsf{PPRF.Eval}(\mathsf{k}_j, i) + (x \cdot y_j, \chi \cdot y_j) \cdot \Delta_{i,\alpha_j},$$

where $\Delta_{i,\alpha_j} = 1$ iff $i = \alpha_j$. Let $(v_{S,2i}^j, v_{S,2i+1}^j) \leftarrow \mathsf{PPRF.Eval}(\mathsf{k}_j, i)$ for $j \in \{1, \ldots, t\}, i \in [N]$. Then, for $i \in [N], j \in \{1, \ldots, t\}$ we have $(v_{R,2i}^j, v_{R,2i+1}^j) = -(v_{S,2i}^j, v_{S,2i+1}^j) + (x \cdot y_j, \chi \cdot y_j) \cdot \Delta_{i,\alpha_j}$. This yields the required, as for each $j \in \{1, \ldots, t\}$ we have $\sum_{i=0}^{N-1} \tau_i \cdot y_j \cdot \Delta_{i,\alpha_j} = \tau_{\alpha_j} \cdot y_j$.

Protocol $\Pi_{\text{m-Gen}}$:

PARAMETERS: $1^\lambda, N = 2^\ell, t, p, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N]$, key space $\{0,1\}^\lambda$ and range $(\mathbb{F}_{p^r})^2$.

INPUTS:

- R inputs distinct indices $\alpha_1, \ldots, \alpha_t \in [N]$ and weights $y_1, \ldots, y_t \in \mathbb{F}_p^*$. We define $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\}$ and $\vec{y} = (y_1, \ldots, y_t) \in (\mathbb{F}_p^*)^t$.
- S inputs $x \in \mathbb{F}_{p^r}$.

PROTOCOL:

1. S picks $\vec{\beta}, \vec{b} \xleftarrow{\$} \mathbb{F}_{p^r}^t$ and $\chi \xleftarrow{\$} \mathbb{F}_{p^r}$.
2. R and S call $\mathcal{F}_{\text{g-rev-VOLE}}(t, p, r)$ on respective inputs $\vec{y}$ and $((\vec{\beta}, \chi), (\vec{b}, x))$. R receives $(\vec{\gamma}, \vec{c}) \in \mathbb{F}_{p^r}^t \times \mathbb{F}_{p^r}^t$.
3. R and S call $\mathcal{F}_{\text{m-PPRF}}(1^\lambda, N, p, r)$ t times on respective inputs α_j and (β_j, b_j). R receives k_j^* and (ω_j, w_j) for each $j \in \{1, \ldots, t\}$. If any of the runs is not successful, R receives *abort* from the functionality.
4. R samples $\tau, \tau_0, \ldots, \tau_{N-1} \xleftarrow{\$} \mathbb{F}_{p^r}$ and sends these to S.
5. S computes $(v_{S,2i}^j, v_{S,2i+1}^j) \leftarrow \mathsf{PPRF.Eval}(\mathsf{k}_j, i)$ for $i \in [N], j \in \{1, \ldots, t\}$, and sends $X = \chi + \tau \cdot x$ and $V_{S,j} = \sum_{i=0}^{N-1} \tau_i \cdot (v_{S,2i}^j + \tau \cdot v_{S,2i+1}^j)$ for $j \in \{1, \ldots, t\}$ to R.
6. R computes $(v_{R,2i}^j, v_{R,2i+1}^j) \leftarrow \mathsf{PPRF.Eval}'(\mathsf{k}_j^*, i)$ for $i \in [N], j \in \{1, \ldots, t\}$, where $\mathsf{PPRF.Eval}'$ is an algorithm that outputs $(\omega_j, w_j) + (\gamma_j, c_j)$ on input $(\mathsf{k}_j^*, \alpha_j)$, and $-\mathsf{PPRF.Eval}(\mathsf{k}_j^*, i)$ else.
7. R checks if $V_{S,j} + \sum_{i=0}^{N-1} \tau_i \cdot (v_{R,2i}^j + \tau \cdot v_{R,2i+1}^j) = X \cdot \tau_{\alpha_j} \cdot y_j$ for $j \in \{1, \ldots, t\}$. If any of these checks fail or R received *abort* from $\mathcal{F}_{\text{m-PPRF}}$ in step 3, R aborts. Otherwise, R sends *ok* to $\mathcal{F}_{\text{m-PPRF}}$ and outputs $\mathsf{k}_0 = (\{\mathsf{k}_j^*, z_j\}_j, \mathcal{S}, \vec{y})$.

Figure 21: Protocol for malicious distributed setup of t puncturable PRFs

Security against malicious sender: On input $((\vec{\beta}, \chi), (\vec{b}, x))$ to the functionality $\mathcal{F}_{\text{g-rev-VOLE}}$ and $(\hat{\beta}_j, \hat{b}_j), I_j, K_j^*$ to $\mathcal{F}_{\text{m-PPRF}}$ by $\mathcal{A}$, the simulator draws challenges $\tau, \tau_0, \ldots, \tau_{N-1} \leftarrow \mathbb{F}_{p^r}$ uniformly at random, and forwards all to $\mathcal{A}$. On input X and $V_{S,1}, \ldots, V_{S,t}$ by $\mathcal{A}$, the simulator proceeds as follows:

1. Set $\delta_j = \hat{\beta}_j - \beta_j$, $d_j = \hat{b}_j - b$ for $j \in \{1, \ldots, t\}$. If there exists a $j \in \{1, \ldots, t\}$ with $d_j \neq 0$, but $\delta_j - \tau \cdot d_j = 0$, abort.
2. For $i \in [N], j \in \{1, \ldots, t\}$: Compute

$$(v_{S,2i}^j, v_{S,2i+1}^j) \leftarrow \mathsf{Eval}^*(K_i^*, I, i)$$

and $\hat{V}_{S,j} = \sum_{i=0}^{N-1} \tau_i \cdot (v_{S,2i}^j + \tau \cdot v_{S,2i+1}^j)$. For each $j \in \{1, \ldots, t\}$ with $d_j \neq 0$ find α_j^*, such that

$$\hat{V}_{S,j} - V_{S,j} = \tau_{\alpha_j^*} \cdot (\delta_j - \tau \cdot d_j), \tag{2}$$

where $\tau_{\alpha_j^*}$ corresponds to the α_j^*-th coefficient chosen by the simulator. If such an α_j^* does not exist or is not unique, abort.

(3) For $j \in \{1, \ldots, t\}$ set

$$\hat{I}_j = \begin{cases} I_j \cap \{\alpha_j^*\} & \text{if } d_j \neq 0, \\ I_j & \text{else} \end{cases}.$$

(4) Parse $K_j^* = \{(K_{j,0}, K_{j,1})_{i=1}^{\ell+1}\}$. Set

$$\hat{K}_{j,0}^{\ell+1} = \begin{cases} K_{j,0}^{\ell+1} - (\delta_j, d_j) & \text{if } d_j \neq 0, \\ K_{j,0}^{\ell+1} & \text{else} \end{cases}.$$

Define $\hat{K}_j^\star = \{(K_{j,0}^i, K_{j,1}^i)_{i=1}^{\ell}\} \cup \{(\hat{K}_{j,0}^{\ell+1}, K_{j,1}^{\ell+1})\}$.

(5) Input $x, \hat{I}_1, \ldots, \hat{I}_t$ and $\hat{K}_1^*, \ldots, \hat{K}_t^*$ to the functionality $\mathcal{F}_{\text{m-Gen}}$. If the functionality does not reply *success*, abort.

We have to show that the probability of the simulation aborting is negligibly close to the probability that the receiver would have aborted in a real execution.

As τ is chosen at random from $\mathbb{F}_{p^r}$ by the simulator, by a union bound over $j \in \{1, \ldots, t\}$, the probability that there exists a $j \in \{1, \ldots, t\}$ with $\delta_j \neq 0$ and $\delta_j - \tau \cdot d_j = 0$ is upper bounded by t/p^r.

Next, we show that passing the verification check, corresponds to guessing the input α_j of the receiver for all j with $d_j \neq 0$.

For $j \in \{1, \ldots, t\}$ it holds $(\omega_j, w_j) = -\mathsf{PPRF.Eval}^*(K_j^*, I_j, \alpha_j) + (\beta_j, b_j) + (\delta_j, d_j)$. Therefore, for the value $v_R^j \in [2N]$ computed by an honest receiver during the real protocol execution, we have

$$(v_{R,2i}^j, v_{R,2i+1}^j) = -\mathsf{PPRF.Eval}(\mathsf{k}_j, i) + ((x, \chi) \cdot y_j + (\delta_j, d_j)) \cdot \Delta_{i,\alpha_j},$$

for all $i \in [N], j \in \{1, \ldots, t\}$. As we have $\sum_{i=1}^N \tau_j(\delta_j + \tau \cdot d_j)\Delta_{i,\alpha_j} = \tau_{\alpha_j}\delta_j + \tau \cdot d_j$, we see that the sender passes the check in the real execution of the protocol, if and only if he provides $V_{S_1}, \ldots, V_{S_t}$, such that $V_{S,j} = \hat{V}_{S,j} - \tau_{\alpha_j} \cdot (\delta_j - \tau \cdot d_j)$ for all $j \in \{1, \ldots, t\}$ with $d_j \neq 0$. Thus, if the sender guessed α_j^* such that $\alpha_j = \alpha_j^*$ for all such j, then the real execution check would have passed. On the other hand, if Equation 2 does not have a solution, the sender would have failed the real world check independent of the choices $\alpha_1, \ldots, \alpha_t$ of the receiver. The coefficients $\tau_0, \ldots, \tau_{N-1}$ provided by the simulator are distinct except with probability at most N/p^r, therefore Equation 2 has a unique solution for each j with $d_j \neq 0$ except with negligible probability.

Finally, by the definition of Eval^*, we have $-\mathsf{PPRF.Eval}^*(K_j^*, I_j, \alpha_j) + (\beta_j, b_j) + (\delta_j, d_j) = -\mathsf{PPRF.Eval}^*(\hat{K}_j^*, I_j, \alpha_j) + (\beta_j, b_j)$, therefore the output to R corresponds to the output in the real protocol execution. □

Note that for $p = 2$, we extract x by finding $j \in \{1, \ldots, t\}$ such that $\chi_j + c \cdot x_j = X$ and set $d_j = x_j - x$.

F.4 Proof of PCG Protocol for VOLE

Definition F.4 (Dual-LPN assumption with static leakage). Let $H \in \mathbb{F}_p^{N \times n}$, and consider the following game $G_b(\lambda)$ with a p.p.t. adversary $\mathcal{A}$, parameterized by a bit b and the security parameter λ:

- Sample $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $S = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$.
- $\mathcal{A}$ sends t sets $I_1, \ldots, I_t \subseteq [n]$.
- If $\alpha_j \in I_j$ for all $j \in \{1, \ldots, t\}$ then send *success* to $\mathcal{A}$, otherwise abort.

Functionality $\mathcal{F}_{\mathsf{sVOLE}}$:

Parameters: $n, p, r \in \mathbb{N}$.

Functionality:

- Sample $\vec{u} \xleftarrow{\$} \mathbb{F}_p^n, \vec{v} \xleftarrow{\$} \mathbb{F}_{p^r}^n, x \xleftarrow{\$} \mathbb{F}_{p^r}$, and let $\vec{w} = \vec{u}x + \vec{v}$
  - If $\mathsf{R_{vole}}$ is corrupt: receive $x, \vec{w}$ from $\mathcal{A}$ and recompute $\vec{v} = \vec{w} - \vec{u}x$
  - If $\mathsf{S_{vole}}$ is corrupt: receive $\vec{u}, \vec{v}$ from $\mathcal{A}$ and recompute $\vec{w} = \vec{u}x + \vec{v}$
- Output $(x, \vec{w})$ to $\mathsf{R_{vole}}$ and $(\vec{u}, \vec{v})$ to $\mathsf{S_{vole}}$

Figure 22: Functionality for corruptible subfield-VOLE correlated randomness

- If $b = 1$, let $\vec{y} = \vec{e} \cdot H$, otherwise sample $\vec{y} \xleftarrow{\$} \mathbb{F}_p^n$
- Send $\vec{y}$ to $\mathcal{A}$, who then outputs a bit b' (in case of abort, define the output of $\mathcal{A}$ to be $\perp$)

The assumption states that $\Pr[\mathcal{A}^{G_0(\lambda)} = 1] - \Pr[\mathcal{A}^{G_1(\lambda)} = 1]$ is negligible in λ.

Definition F.5 (Dual-LPN assumption with adaptive leakage). Let $H \in \mathbb{F}_p^{N \times n}$, and consider the following game $G_b^{\mathrm{a}}(\lambda)$ with a p.p.t. adversary $\mathcal{A}$, parameterized by a bit b and the security parameter λ:

- Sample $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $S = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$.
- If $b = 1$, let $\vec{y} = \vec{e} \cdot H$, otherwise sample $\vec{y} \xleftarrow{\$} \mathbb{F}_p^n$
- Send $\vec{y}$ to $\mathcal{A}$
- $\mathcal{A}$ responds with t sets $I_1, \ldots, I_t \subseteq [n]$
- If $\alpha_j \in I_j$ for all $j \in \{1, \ldots, t\}$ then send *success* to $\mathcal{A}$, otherwise send *fail*
- $\mathcal{A}$ outputs a bit b'

The assumption states that $\Pr[\mathcal{A}^{G_0^{\mathrm{a}}(\lambda)} = 1] - \Pr[\mathcal{A}^{G_1^{\mathrm{a}}(\lambda)} = 1]$ is negligible in λ.

Theorem 6.3. *Let* PPRF *be a t-puncturable PRF, and suppose that* $(\mathcal{HW}_t, \mathrm{C}, \mathbb{F}_p)$-dual-LPN$(N, n)$ *with static leakage holds. The protocol in Fig. 23 securely realizes the functionality* $\mathcal{F}_{\mathsf{sVOLE}}$.

Proof. The case where both parties are corrupted is straightforward. When neither party is corrupted, by inspection, the outputs of the honest parties satisfy $\vec{w} = \vec{u}x + \vec{v}$, so we just need to show that these values are uniform subject to this constraint. The VOLE sender's $\vec{u}$ output is pseudorandom under the dual-LPN assumption, since we have $\vec{u} = \vec{e} \cdot H$, and the receiver's x is uniformly random. Finally the sender's $\vec{v} = \vec{v}_1 \cdot H$ value is pseudorandom, since $\vec{v}_1$ consists of pseudorandom PPRF outputs, and the matrix H has full rank.

$\mathsf{S_{vole}}$ *is corrupted.* The simulator Sim_S proceeds as follows.

(1) Sim_S receives the malicious $\mathsf{S_{vole}}$'s inputs to $\mathcal{F}_{\text{m-Gen}}$, $\vec{y}, S = (\alpha_1, \ldots, \alpha_t)$, and defines the weight $\leq t$ error vector $\vec{e}$.

(2) Sim_S samples a PPRF key $\mathsf{k_{pprf}} \xleftarrow{\$} \{0,1\}^\lambda$ and computes the punctured key $\mathsf{k_{pprf}^*} = \mathsf{PPRF.Puncture}(\mathsf{k_{pprf}}, S)$.

(3) It sends $\mathsf{k_{pprf}^*}$ to $\mathsf{S_{vole}}$, along with random $z_j \xleftarrow{\$} \mathbb{F}_{p^r}$, for $j \in \{1, \ldots, t\}$.

Protocol Π_{sVOLE}:

PARAMETERS: $1^\lambda, N = 2^\ell, n, p, r, t \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N] = \{0,1\}^\ell$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}_{p^r}$, supporting verification of malicious keys.
$H \in \mathbb{F}_p^{N\times n}$ is a matrix for which dual-LPN is hard.

PROTOCOL:

(1) $\mathsf{S}_{\mathsf{vole}}$ samples a weight-t vector $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $S = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$, and $y_i = e_{\alpha_i} \in \mathbb{F}_p^*$.
(2) $\mathsf{R}_{\mathsf{vole}}$ samples $x \xleftarrow{\$} \mathbb{F}_{p^r}$ and $\mathsf{k}_{\mathsf{pprf}} \xleftarrow{\$} \{0,1\}^\lambda$.
(3) $\mathsf{R}_{\mathsf{vole}}$ and $\mathsf{S}_{\mathsf{vole}}$ call $\mathcal{F}_{\mathsf{m\text{-}Gen}}$ (with the roles of sender and receiver reversed) on inputs $(\vec{y}, \alpha_1, \ldots, \alpha_t)$ and $(\mathsf{k}_{\mathsf{pprf}}, x)$.
(4) $\mathsf{S}_{\mathsf{vole}}$ receives a punctured key k_j^* and value $z_j \in \mathbb{F}_{p^r}$, for $j \in \{1, \ldots, t\}$.
(5) $\mathsf{R}_{\mathsf{vole}}$ receives the master keys $\mathsf{k}_1, \ldots, \mathsf{k}_t$.
(6) $\mathsf{S}_{\mathsf{vole}}$ outputs $(\vec{u}, \vec{v}) \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(0, \{\mathsf{k}_j^*, z_j\}_{j=1}^t, \vec{e})$
(7) $\mathsf{R}_{\mathsf{vole}}$ outputs $(x, \vec{w}) \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(1, \{\mathsf{k}_1, \ldots, \mathsf{k}_t\}, x)$

Figure 23: Protocol for realizing random subfield VOLE of length n with malicious security

(4) Sim_S computes $\vec{u}, \vec{w} = G_{\mathsf{sVOLE}}.\mathsf{Expand}(0, \mathsf{k}_{\mathsf{pprf}}^*, \{z_j\}_{j=1}^t, \vec{e})$ and then sends $\vec{u}, \vec{w}$ to $\mathcal{F}_{\mathsf{sVOLE}}$.

Notice that the only difference between the simulation and the real execution is the way the z_j values are computed. In the protocol, z_j masks the sender's inputs with PPRF evaluations at the punctured points, whereas in the simulation z_j is random. These two views are indistinguishable, by the security of PPRF; any adversary that distinguishes the two executions can be used to win the PPRF selective security game, Exp-s-pPRF, with exactly the same advantage.

$\mathsf{R}_{\mathsf{vole}}$ *is corrupted.*

(1) Sim_R receives from $\mathsf{R}_{\mathsf{vole}}$ $x \in \mathbb{F}_{p^r}$, the subsets $I_1, \ldots, I_t$ and keys $K_1^*, \ldots, K_t^* \in \mathcal{K}$
(2) Sample $\alpha_1, \ldots, \alpha_t \xleftarrow{\$} [N]$, and for each $j \in \{1, \ldots, t\}$, check that (i) $\alpha_j \in I_j$, and (ii) $\mathsf{Ver}(K_j^*, I_j) = 1$. If any check fails, abort.
(3) Sim_R computes the PPRF outputs $\vec{v}_1$ using $\mathsf{Eval}^*(K_j^*, I_j, x)$, for all x, j, and uses these to compute $\vec{w} = \vec{v}_1 \cdot H$ that is sent to $\mathcal{F}_{\mathsf{sVOLE}}$, along with x.

Notice that the probability of abort is identical in both executions, since Sim_R samples a noise vector just as in the real protocol. Also, the outputs of the corrupt VOLE receiver, computed by Sim_R, are identically distributed to those in the protocol. The only difference between the two executions is the way the honest sender's outputs $(\vec{u}, \vec{v})$ are computed; here, we rely on the leaky variant of dual-LPN to argue that $\vec{u}$ is pseudorandom.

In particular, we show that any distinguisher $\mathcal{D}$, who distinguishes the real and ideal executions, can be used against the dual-LPN assumption with static leakage. We construct an adversary for game G as follows: Invoke $\mathcal{D}$ with an execution of Π_{sVOLE}, and, running Sim_R, receive the adversary's guesses $I_1, \ldots, I_t$. Instead of sampling α_j as Sim_R does, forward the guesses to game G. If G aborts, send *abort* to $\mathcal{D}$, otherwise, continue running Sim_R. At the end of the execution (if it did not abort) send to $\mathcal{D}$ the honest sender's output $(\vec{u}, \vec{v})$, where $\vec{u}$ is set to the $\vec{y}$ vector received from G, and $\vec{v} = \vec{w} - \vec{u}x$. Output whatever $\mathcal{D}$ outputs.

When $b = 1$, the view of $\mathcal{D}$ is as in the real protocol, whereas when $b = 0$ it is exactly as in the simulation, hence, our advantage against the game G is exactly the same as the advantage of $\mathcal{D}$ against the protocol. □

F.5 4-Round Random OT PCG Protocol with Malicious Security

Given the PCG protocol for the subfield VOLE correlation, we can easily convert this a PCG protocol for random oblivious transfer using a correlation robust hash function. To perform 1-out-of-2 OT, the parties run subfield VOLE over $\mathbb{F}_{2^\lambda}$, where the OT receiver, who acts as VOLE sender, obtains $\vec{u} \in \mathbb{F}_2^n$ and $\vec{v} = \vec{w} + \vec{u}x \in \mathbb{F}_{2^\lambda}^n$, while the OT sender / VOLE receiver gets $x \in \mathbb{F}_{2^\lambda}, \vec{w} \in \mathbb{F}_{2^\lambda}^n$. This can be seen as a correlated OT, where the receiver has choice bits u_i and messages v_i, which equal either w_i or $w_i + x$. To convert these to random OTs, the parties use a hash function H, and output respectively

$$(u_i, \mathsf{H}(i, v_i)), \quad (\mathsf{H}(i, w_i), \mathsf{H}(i, w_i + x))$$

We require the hash function to have a suitable correlation robustness property, namely, for any choices of w_i, the pairs $(w_i, \mathsf{H}(i, w_i + x))_i$ are indistinguishable from pairs $(w_i, U)_i$, where U is the uniform distribution [17, 37]. This then gives a protocol that realizes the corruptible random OT functionality, where corrupt parties may influence their random outputs, in the $\mathcal{F}_{\mathsf{sVOLE}}$-hybrid model.

Notice that, as observed in [33], in order to prove security the index i must be included as an input to H, since otherwise a malicious receiver can break security by choosing its outputs of the corruptible $\mathcal{F}_{\mathsf{sVOLE}}$ functionality (where it is VOLE sender).[3]

THEOREM F.6. *Suppose that there exists an $\mathbb{F}_p$-correlation-robust hash function, and $(\mathcal{HW}_t, \mathsf{C}, \mathbb{F}_p)$-dual-LPN$(N, n)$ with static leakage holds for $N = O(n)$ and $t = o(n/(\lambda \log n)$. Then there exists a 4-message, maliciously secure PCG protocol for the random 1-out-of-p OT correlation, which makes $o(n)$ parallel calls to an oblivious transfer functionality and communicates $o(n)$ bits.*

F.6 2-Round OT Extension and Silent NISC with Malicious Security

We now show how to compress the above protocols down to just two rounds, by applying the Fiat-Shamir heuristic. Additionally, under a slightly stronger version of the dual-LPN assumption with one bit of *adaptive* leakage (see Definition F.5 in Section F.4), we

[3] In practice, however, we do not know of an attack on our concrete protocol if this is omitted, since a malicious VOLE sender cannot actually choose its $\vec{v}$ outputs, this is only needed for the security proof.

can convert the random OTs/VOLEs into ones on chosen inputs in parallel with the setup messages, This gives a two-round OT extension protocol with malicious security.

First, observe that the only interaction in the malicious secure protocol for distributing t puncturable PRF keys (Fig. 21 in Appendix F.3), besides the parallel calls to $\mathcal{F}_{\text{m-PPRF}}$ and $\mathcal{F}_{\text{g-rev-VOLE}}$, is the random challenges $(\tau, \tau_0, \ldots, \tau_{N-1})$ from the receiver, and response from the sender. Using the Fiat-Shamir heuristic in the random oracle model, the sender can instead compute the challenges by hashing the transcript. From Theorem 6.1, $\mathcal{F}_{\text{m-PPRF}}$ can be realized with $o(n)$ parallel calls to OT; also, note that $\mathcal{F}_{\text{g-rev-VOLE}}$ can be efficiently realized using any semi-homomorphic encryption scheme which supports zero-knowledge proofs of knowledge for ciphertext generation and homomorphic multiplication. For instance, [12] shows how to instantiate this under the Paillier or LWE assumption. In the random oracle model, the zero-knowledge proofs can be made non-interactive, leading to a two round protocol overall.

Theorem F.7. *Suppose that* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(N, n)$ *with static leakage holds, where* $N = O(n)$ *and* $t = o(n/(\lambda \log n)$*, and a semi-homomorphic encryption scheme exists. Then in the random oracle model, there is a non-interactive, maliciously secure PCG protocol for the subfield VOLE correlation, which makes* $o(n)$ *parallel calls to an oblivious transfer functionality and communicates* $o(n)$ *bits.*

Additionally assuming an $\mathbb{F}_p$*-correlation robust hash function, there is a non-interactive PCG protocol for random 1-out-of-p OT with the same complexity.*

To obtain silent NISC for the OT functionality, which implies two-round OT extension on chosen inputs, we use the observation (as in our semi-honest protocol) that the receiver and sender can derandomize their inputs in parallel with their protocol messages. In the malicious setting, this requires an *adaptive* variant of the dual-LPN with leakage assumption, since a corrupt sender can see the masked receiver's inputs before attempting to guess a few of the LPN error positions. We present the complete protocol for 1-out-of-2 OT in Figure 24.

Theorem F.8. *Suppose that there exists a correlation-robust hash function, and* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_2)$-dual-LPN$(N, n)$ *with adaptive leakage holds for* $N = O(n)$ *and* $t = o(n/(\lambda \log n)$*. Then in the random oracle model, there exists a maliciously secure 2-message protocol for realizing n 1-out-of-2 oblivious transfers, which makes* $o(n)$ *parallel calls to an oblivious transfer functionality and communicates* $o(n)$ *bits.*

The above theorem also extends to 1-out-of-p OT for prime $p > 2$, by additionally assuming a semi-homomorphic encryption scheme as in Theorem F.7.

Protocol $\Pi_{\text{m-OT}}$

Parameters: $1^\lambda, n, N = 2^\ell, t, 2, r \in \mathbb{N}$. PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{2^r}$, supporting verification of malicious keys. $H \in \mathbb{F}_2^{N \times n}$ is a matrix for which dual-LPN is hard. $\mathcal{D}_{t,N}$ is a weight-t error distribution over $\mathbb{F}_2^N$. RO is a random oracle with output space $(\mathbb{F}_{2^r})^{N+1}$. H is a correlation robust hash function.

- R inputs n field elements $(s_i)_{i \le n} \in \mathbb{F}_2^n$.
- S inputs n length-2 vectors $(\vec{m}_i)_{i \le n}$ where each $\vec{m}_i$ is over $(\{0,1\}^\lambda)^2$.

Protocol:

(1) R picks $\vec{e} \xleftarrow{\$} \mathcal{D}_{t,N}$. Let $\mathcal{S} = \{\alpha_1, \ldots, \alpha_t\} \in [N]^t$ be the sorted indices of non-zero entries in $\vec{e}$. R computes the first part $\vec{u}$ of $G_{\text{sVOLE}}.\text{Expand}(0, \{\mathsf{k}_j^*, z_j\}_{j=1}^t, \vec{e})$ (note that $\vec{u}$ is computed as $\vec{\mu} \cdot H$ where $\vec{\mu}$ depends solely on $\vec{e}$). R sets $\vec{t} \leftarrow \vec{u} + \vec{s}$.

(2) R sends the first message for the protocol $\Pi_{\text{m-Gen}}$ with input $(\alpha_1, \ldots, \alpha_t)$. Simultaneously to the first message, R sends $\vec{t}$ to S. Let m_{R} denote the accumulation of receiver messages.

(3) S samples $x \xleftarrow{\$} \mathbb{F}_{2^r}$ and $\mathsf{k}_j \xleftarrow{\$} \{0,1\}^\lambda$ for $j \in \{1, \ldots, t\}$, and uses $(x, \mathsf{k}_1, \ldots, \mathsf{k}_t)$ as input to $\Pi_{\text{m-Gen}}$. Let m_{S} denote the corresponding accumulation of messages of S in the protocol execution.
   - To replace the second message of R, S calls $\tau, \tau_1, \ldots, \tau_N \leftarrow \text{RO}(m_{\mathsf{R}}, m_{\mathsf{S}})$ and computes the last message of $\Pi_{\text{m-Gen}}$ with these challenges.
   - S computes $(x, \vec{w}') \leftarrow G_{\text{sVOLE}}.\text{Expand}(1, \mathsf{k})$, where $\mathsf{k} = \{\mathsf{k}_1, \ldots, \mathsf{k}_t\}$.
   - S computes $w_{i,j} \leftarrow \mathsf{H}(i, w'_i - j \cdot x)$ for $i \in \{1, \ldots, n\}, j \in \{0,1\}$,
   - S sets $m'_{i,j} \leftarrow m_{i,j} + w_{i,t_i - j}$ for $i \in \{1, \ldots, n\}, j \in \{0,1\}$.
   - S sends $m_{\mathsf{S}}, X, V_{S,1}, \ldots, V_{S,j}, \{m'_{i,j}\}_{i \le n, j \le 2}$ to R.

(4) R receives m_{S} containing k_j^*, (ζ_j, z_j) for each $j \in \{1, \ldots, t\}$ and further $X, V_{S,1}, \ldots, V_{S,j}$, and $\{m'_{i,j}\}_{i \in \{1,\ldots,n\}, j \in \{1,2\}}$.
   - R verifies all checks in $\Pi_{\text{m-Gen}}$ with $\tau, \tau_1, \ldots, \tau_N = \text{RO}(m_{\mathsf{R}}, m_{\mathsf{S}})$. If any fails, abort.
   - R computes the second message $\vec{v}'$ of $G_{\text{sVOLE}}.\text{Expand}(0, \{\mathsf{k}_j^*, z_j\}_{j=1}^t, \vec{e})$.
   - R computes $v_i \leftarrow \mathsf{H}(i, v'_i)$ for $i \in \{1, \ldots, n\}$.
   - R outputs $(m'_{i,s_i} - v_i)_{i \le n}$.

Figure 24: Two-Round 1-out-of-2 OT Extension with Malicious Security