MPC-Friendly Symmetric Cryptography from Alternating Moduli: Candidates, Protocols, and Applications

Anonymous Crypto 2021 Submission

Abstract. We study new candidates for symmetric cryptographic primitives that leverage alternation between linear functions over $\mathbb{Z}_2$ and $\mathbb{Z}_3$ to support fast protocols for secure multiparty computation (MPC). This continues the study of weak pseudorandom functions of this kind initiated by Boneh et al. (TCC 2018) and Cheon et al. (ePrint 2020).

We make the following contributions.

- **Candidates.** We propose new designs of symmetric primitives based on alternating moduli. These include candidate one-way functions, pseudorandom generators, and weak pseudorandom functions in which the key, the input and the output are all over $\mathbb{Z}_2$. We propose concrete parameters based on cryptanalysis.
- **Protocols.** We provide a unified approach for securely evaluating modulus-alternating primitives in different MPC models. For the original candidate of Boneh et al., our protocols obtain roughly 2x improvement in all performance measures. We report efficiency benchmarks of an optimized implementation.
- **Applications.** We showcase the usefulness of our candidates for a variety of applications. This includes short "Picnic-style" signature schemes, as well as protocols for oblivious pseudorandom functions, hierarchical key derivation, and distributed key generation for function secret sharing.

1 Introduction

Symmetric-key primitives like one-way functions (OWFs) [57], pseudorandom generators (PRGs) [14, 71], and pseudo-random functions (PRFs) [43] are deployed in innumerable settings, and serve as fundamental building blocks of modern cryptography. While traditional use cases primarily considered settings where the function evaluation was done by a single party, many applications (recently also arising in the context of cryptocurrencies) require evaluation in a distributed fashion to avoid single points of failure. This motivates the study of secure multiparty computation (MPC) protocols for evaluating such symmetric-key primitives in a setting where inputs, outputs, and keys are secret-shared or distributed between two or more parties.

Towards this goal, a long line of work [34, 63, 69] has made substantial progress on concretely efficient MPC protocols for distributing the computation of symmetric primitives, such as AES or SHA-256, which are widely used in practice. Unfortunately, the constructions themselves were not designed with distributed evaluation in mind, and are often clunky to optimize for. More recent work (see [2–4, 15, 45] and references therein) has therefore proposed to start from scratch by designing *MPC-friendly* primitives from the ground up. In this work, we continue this line of research by proposing a new suite of *simple* MPC-friendly candidate designs for a number of symmetric primitives.

Our MPC setting. We focus on the *semi-honest* setting of security for simplicity. This is considered adequate in many cases. In particular, it suffices for the construction of signature schemes via an "MPC-in-the-head" technique [27, 48]. While recent general techniques from the literature [16, 26] can be used to extend some of our protocols to the malicious security model with a low amortized cost, we leave such an extension to future work. We consider protocols for both two parties (2PC) and multiple parties, both with and without an honest majority assumption, and both with and without preprocessing. In the following, we consider by default the setting of (semi-honest) 2PC with preprocessing. However, our contributions apply to the other settings as well.

Efficiency metrics for MPC. Concretely efficient MPC protocols can be divided into two broad categories: protocols based on *garbled circuits* [72] and protocols based on *linear secret sharing* [11, 28, 44]. Protocols based on garbled circuits have low round complexity but their communication cost will be prohibitively high for our purposes. We will therefore focus on protocols based on secret sharing. Roughly speaking, the complexity of evaluating a given function f using such protocols is determined by the *size* and the *depth* of a *circuit* C that evaluates f. Here we assume that C is comprised of atomic gates of two kinds: *linear gates* (computing modular addition or multiplication by a public value) and MPC-friendly *nonlinear gates* that are supported by efficient subprotocols. A typical example for a nonlinear gate is modular multiplication of two secret values. Given such a representation for f, the *communication* cost of an MPC protocol for f scale linearly with the *size* of C, namely the number of gates weighted by the "MPC cost" of each gate, whereas the *round complexity* scales linearly with the *depth* of C, namely the of gates on a longest input-output path. Since linear gates do not require any interaction, they do not count towards the size or the depth. We use the term "nonlinear size" and "nonlinear depth" to refer to the size and the depth when excluding linear gates.

Our design criteria. The above efficiency metrics for MPC are quite crude, since not all kinds of nonlinear gates are the same. However, they still serve as a good intuitive guideline for the design of MPC-friendly primitives. More concretely, we would like to design primitives with the following goals in mind.

- *Low nonlinear depth.* Minimizing round complexity calls for minimizing nonlinear depth. Unfortunately, constructions like AES or even MPC-friendly ones such as LowMC [2] have quite a high nonlinear depth, which leads to high-latency protocols when using the secret-sharing approach.

- *Small nonlinear size.* For keeping the communication complexity low, we would like to minimize the number of nonlinear gates and make them as "small" and "MPC-friendly" as possible.
- *High algebraic degree.* Security of block ciphers and (weak) PRFs provably requires high algebraic degree. While there are low-degree implementations of weaker primitives such as OWF and PRG [5, 42, 59], these typically come at the price of bigger input size and higher nonlinear size [33].
- *Simplicity.* A simple design is almost always easier to implement and prone to fewer errors and attacks. This is particularly valuable since a substantial amount of work has previously gone into implementations that resist timing and cache side-channels. Simple constructions are also easier to reason about and cryptanalyze, which builds confidence in their security, and may serve as interesting objects of study from a theory perspective [1, 42, 61].

The alternating moduli paradigm. The above design goals seem inherently at odds with each other. How can "high algebraic degree" co-exist with "small gates" and "low nonlinear depth"? Towards settling this apparent paradox, a new design paradigm was recently proposed by Boneh et al. [15] and further analyzed by Cheon et al. [32]. The idea is to break the computation into two or more parts, where each part includes a linear function over a *different modulus*. The simplest choice of moduli, which also seems to lead to the best efficiency, is 2 and 3.

Boneh et al. [15] proposed a weak PRF[1] (wPRF) candidate with the following simple description: the input x is a vector over $\mathbb{Z}_2$ and the secret key specifies a matrix $\mathbf{K}$ over $\mathbb{Z}_2$. The PRF first computes the matrix-vector product $\mathbf{KX}$ over $\mathbb{Z}_2$, then interprets the result as a vector over $\mathbb{Z}_3$ in the natural way, and finally applies a public, compressive linear mapping over $\mathbb{Z}_3$ to obtain an output vector y over $\mathbb{Z}_3$.

The above mapping from x and K to y has two nonlinear steps: The first is the matrix-vector product over $\mathbb{Z}_2$, whose cost can be reduced when the matrix K has a special form. The second is a *conversion* of a mod-2 vector to a mod-3 vector, which consists of small parallel (finite-size) nonlinear gates. Overall, the nonlinear depth is 2. Why is this a high-degree function? Viewing both the input and the (binary representation of) the output as vectors over $\mathbb{Z}_2$, high degree over $\mathbb{Z}_2$ comes from the final linear mapping over $\mathbb{Z}_3$. Viewing the input as a vector over $\mathbb{Z}_3$, high degree comes from the linear mapping over $\mathbb{Z}_2$ defined by the key. Despite its simplicity, the design can be conjectured to have a good level of security with small input and key size (say, 256 bits for 128-bit security). It mostly resisted the initial cryptanalysis, where attacks found in [32] require a very high sample complexity and are quite easy to circumvent by slightly modifying the design (as suggested in [32]).

A primary motivation for the alternating moduli paradigm was its MPC-friendliness. Indeed, several MPC protocols were proposed in [15]. These pro-

[1] A weak PRF is one whose security only holds when evaluated on *random* inputs. In many applications of strong PRF, a weak PRF can be used instead by first applying a hash function (modeled as a random oracle) to the input

tocols demonstrate significant efficiency advantages over earlier MPC-friendly designs, mainly in the setting of 2PC with preprocessing or 3-party computation with an honest majority.

Another, very different, motivation is the goal of identifying simple function classes that are "hard to learn." Indeed, the conjectures from [15] imply hardness of learning results for low complexity functions such as (depth-2) ACC^0 circuits, sparse $\mathbb{Z}_3$ polynomials, or width-3 branching programs. These conjectures are also of interest outside the field of cryptography [29, 30, 40, 52], which further motivates cryptanalysis efforts.

Remaining challenges. The initial works of [15, 32] have only scratched the surface of the kind of questions one may ask.

- What about simpler symmetric primitives such as OWFs and PRGs? MPC protocols for these primitives are motivated by many applications, including Picnic-style post-quantum digital signatures [27, 53] and lightweight distributed key generation for function secret sharing.
- Are there similar wPRF candidates where the input, output, and key are all over $\mathbb{Z}_2$? This too is motivated by natural applications.
- Can the concrete MPC protocols given in [15] be further improved? Can the preprocessing be realized at a low amortized cost? This motivates an additional design criterion: "PCG-friendliness," leveraging recent advances in pseudorandom correlation generators [21, 24, 70].

1.1 Our Contribution

Motivated by the above questions, we make the following contributions.

1.1.1 New candidate constructions.

We introduce several candidate constructions for OWF, PRG, and (weak) PRF, all based on alternating between linear mappings over $\mathbb{Z}_2$ and $\mathbb{Z}_3$.

- **Candidate OWF.** We expand on the general structure of the $(2,3)$-wPRF candidate from [15] to construct a candidate OWF. Recall that the wPRF candidate computes $\mathbf{B}(\mathbf{K}x)$ where $\mathbf{K}$ is the secret key (over $\mathbb{Z}_2$) and $\mathbf{B}$ is a compressive $\mathbb{Z}_3$ linear mapping. For our $(2,3)$-OWF candidate, we replace the secret key matrix with another randomly sampled (expanding) public matrix $\mathbf{A}$. Specifically, given $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ and $\mathbf{B} \in \mathbb{Z}_3^{t\times m}$ where $m \geq n, t$, our OWF candidate is defined as $\mathsf{F}(x) = \mathbf{B}(\mathbf{A}x)$ where $\mathbf{A}x$ is first reinterpreted as a 0/1 vector over $\mathbb{Z}_3$.
- **Candidate wPRF.** The wPRF candidate from [15] had inputs over $\mathbb{Z}_2$ but outputs over $\mathbb{Z}_3$ which is not sufficient for many practical applications. Therefore, here we introduce an LPN-style wPRF candidate where both the input and output are over $\mathbb{Z}_2$. Specifically, given a secret key matrix $\mathbf{K} \in \mathbb{Z}_2^{m\times n}$ and a public compressive mapping $\mathbf{B} \in \mathbb{Z}_2^{t\times m}$, for an input $x \in \mathbb{Z}_2^n$, our LPN-wPRF candidate first computes an intermediate vector

$$w = [(\mathbf{K}x \bmod 2) + (\mathbf{K}x \bmod 3) \bmod 2] \bmod 2$$

where for $\mathbf{K}x \bmod 3$, both $\mathbf{K}$ and x are first reinterpreted over $\mathbb{Z}_3$. Then, the candidate is defined as $\mathsf{F}_{\mathbf{K}}(x) = \mathbf{B}w$. Intuitively, each intermediate vector bit can be thought of as a deterministic Learning-Parity-with-Noise (LPN) instance with a noise rate of $1/3$. The noise is deterministically generated and is dependent on the input x and a specific column of $\mathbf{A}$. A similar candidate was considered in [15] (as their alternate candidate) but it only outputs a single bit (it uses $\mathbf{K} \in \mathbb{Z}_2^{1\times n}$ and outputs the intermediate vector directly). Our candidate generalizes this to multiple output bits. But more importantly, it also does not output the intermediate vector directly and instead first applies an additional compressive linear mapping (using $\mathbf{B}$). We show how this allows our candidate to restrict standard attacks on LPN.

- **Candidate PRG.** We also propose a candidate length-doubling PRG that is similar to our LPN-wPRF. Specifically, we use a public matrix $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ instead of the key for the first linear mappings. It follows the same structure as the LPN-wPRF, by first expanding the input to the intermediate vector w and then applying a contracting $\mathbb{Z}_2$ linear mapping $\mathbf{B}$. Choosing m, i.e., the length of the intermediate vector, to be large enough, we can ensure that the final compressive mapping still results in an output of size $t = 2n$.

1.1.2 Cryptanalysis and implications on parameter choices.

Given that the constructions heavily mix linear operations over $\mathbb{Z}_2$ and $\mathbb{Z}_3$, we will rely on the arguments of Boneh et al. [15], and conjecture that algebraic attacks do not threaten their security. Instead, we will focus of combinatorial attacks and statistical tests.

Our most interesting attack on the candidate OWF reduces the inversion problem to a particular type of subset-sum problem, where addition simultaneously involves operations over $\mathbb{Z}_2$ and $\mathbb{Z}_3$. Thus, we can invert the OWF by applying a variant of recent advanced algorithms for solving subset-sum that are based on the *representation technique* [9, 17, 46]. Compared to a standard meet-in-the-middle approach, this attack forced us to increase the parameters by about 30%.

For the candidate wPRF and PRG constructions, the security analysis is mainly focused on statistical distinguishers that exploit a bias in the output. The strength of such a bias depends on the minimal distance of the code generated by the second linear operation of the construction. As this code is generated at random, we use the probabilistic method (in a similar way it is used to obtain the Gilbert–Varshamov bound for linear codes) to argue that its minimal distance is sufficiently large, except with negligible probability.

In Table 1 we summarize the recommended concrete parameters for our constructions with the goal of obtaining s-bit security. For the $(2,3)$-OWF and $(2,3)$-wPRF constructions we give both aggressive and more conservative parameter sets. Note that the OWF and PRG use the minimal secret input (and output) sizes, while for wPRFs, we use a larger secret. This is a result of different tradeoffs between security and performance. For example, we could have set $n = s$ for the $(2,3)$-wPRF, but cryptanalysis would force setting m to be much

Construction	Parameters (n, m, t)	Comment
$(2,3)$-OWF	$(s, 3.13s, s/\log 3)$	aggressive
	$(s, 3.53s, s/\log 3)$	conservative
$(2,3)$-wPRF	$(2s, 2s, s/\log 3)$	aggressive
	$(2s, 2.5s, s/\log 3)$	conservative
LPN-PRG	$(s, 3s, 2s)$	
LPN-PRF	$(2s, 2s, s)$	

Table 1: Concrete parameters for s-bit security.

larger than $2s$ and result is less efficient protocols. On the other hand, setting $n = 2s$ for the $(2,3)$-OWF would also require doubling the size of the output,[2] once again, degrading efficiency.

Our constructions are new and it is reasonable that some of them would be broken and require updating the parameter sets (even the "conservative" ones). Conversely, if for some of our constructions the more aggressive parameter sets would not be broken, we would gain confidence in their security.

One of the main questions we leave open is how to better exploit the structured matrices used in our constructions in cryptanalysis. This question is particularly interesting for the wPRF constructions where the attacker obtains several samples, and can perhaps utilize the structured matrices to combine their information in more efficient attacks.

1.1.3 Distributed protocols and optimized implementations.

Efficient protocols. We provide efficient protocols for our wPRF candidate constructions in a variety of settings: 2PC with preprocessing, 3PC with one passive corruption, and Oblivious 2PC. For the $(2,3)$-wPRF candidate, our protocols when compared to the ones from [15] in exactly the same setting perform roughly 1.5-5x better in both online communication and preprocessing size. Specifically, in the 2PC setting, our protocol requires 2 rounds, 1536 bits of online communication, and 662 bits of preprocessing. In contrast, the protocol from [15] for the same setting requires 4 rounds, roughly 2600 bits of online communication and roughly 3500 bits of preprocessing. Similarly, our OPRF protocol requires 2 rounds, 641 bits of online communication while the one from [15] requires 4 rounds and roughly 1800 bits of online communication.

The key ingredient which gives our protocols their efficiency is a subprotocol for modulus conversion gates that switch between shares in $\mathbb{Z}_2$ and $\mathbb{Z}_3$. While [15] used OT in their protocols, we use these modulus conversion gates for better efficiency. We note that these conversion gates can also be used to construct efficient distributed protocols for alternate variants of our constructions.

[2] Otherwise, each output would have 2^s preimages and there would be no security advantage.

Distributing the dealer at a low amortized cost. The 2PC protocols presented in [15] rely on trusted preprocessing to generate two kinds of correlated randomness. The first kind, used to securely multiply the input and the key matrix, can be thought of as a standard multiplication triple over a ring. (Using a circulant matrix for the key, this involves a single multiplication in a ring of polynomials over $\mathbb{Z}_2$.) It was also pointed out that using efficient pseudorandom correlation generators (PCGs) for *vector oblivious-linear evaluation* (VOLE) correlations [21, 24, 66], this kind of correlation can be generated at a low amortized cost when the same key is reused with multiple inputs. In fact, using more recent PCGs for independent OLE correlations [22] the latter restriction can be removed, albeit at a considerably higher cost. The second kind of correlated randomness used in [15] is a standard oblivious transfer (OT) correlation, which can also be efficiently generated using either classical [47] or "silent" [24, 70] OT extension. The latter techniques use a PCG for OT to enable fast local generation of many random instances of OT from a pair of short, correlated seeds. However, the main source of improvement of our protocols over the ones from [15] is our use of the *modulus conversion* correlations described above. We show how to generate both kinds of correlations from a standard OT correlation using only a *single* message, where in the $\mathbb{Z}_2 \to \mathbb{Z}_3$ case the (amortized) communication is < 1.38 bits per instance, and in the (less commonly used) $\mathbb{Z}_3 \to \mathbb{Z}_2$ case it is 6 bits per instance. This means that the amortized cost of distributing the dealer in our protocols is typically much lower than the cost of the online protocol that consumes the correlated randomness.

1.1.4 Applications.

Digital signatures. Using the MPC-friendliness of candidates, we can efficiently prove knowledge of an input (e.g., of a OWF input, wPRF key or PRG seed), using proof protocols based on the MPC-in-the-head paradigm [48]. This is the approach taken by many recently designed post-quantum signature schemes [7, 12, 13, 27, 54, 65], as it only requires a secure OWF and hash function, and has opened up the range of hardness assumptions possible for public-key signatures. We present the first public-key signature scheme based on alternating moduli cryptography.

We provide a detailed description of a signature scheme using our OWF candidate, as a modification to the Picnic algorithm [27, 53, 54, 68], a third round candidate in the NIST Post-Quantum Cryptography Standardization Process.[3] We replace the OWF in Picnic (an instance of the LowMC block cipher [2]), update the MPC protocol accordingly, and quantify the resulting signature sizes. We find that signatures sizes are slightly shorter, with signatures at the 128-bit security level (64-bit quantum) having size ranging from 10.3–13.3KB (depending on parameter choice). This shows that OWFs based on alternating moduli are competitive with block-cipher based designs, and we can choose a OWF with an (arguably) simpler mathematical description, without sacrificing performance.

[3] See `https://csrc.nist.gov/projects/post-quantum-cryptography/`.

Oblivious pseudorandom functions. We construct an OPRF protocol that computes our $(2,3)$-wPRF candidate in an oblivious setting. In the multi-input setting (where the key is used for multiple evaluations), our protocol requires only 2 rounds and 641 bits of online communication. Compared to a standard DDH-based OPRF [49, 51], which require 512 bits of communication for 128-bit security, our protocol requires slightly higher communication but has a much better performance. Our implementation shows that our OPRF protocol is faster than a *single* scalar multiplication over the Curve25519 elliptic curve. Consequently, we expect our protocol to be faster than a number of OPRF protocols [41, 50] that are based on algebraic PRFs and require expensive exponentiations. In another direction, recent works [45, 67] construct an OPRF protocol from the Legendre PRF [36]. For 128-bit security and only a *single* output bit, the recent protocol from [67] has online communication cost of 13KB, substantially higher than ours (with 128 output bits), and with a higher computational cost.

Secret output. Our $(2,3)$-wPRF candidate is well suited for applications that have privately held secret-shared inputs but require a public output that is delivered in the clear. However, it is insufficient for applications in which the output of the function needs to itself be kept secret and reused as the input to a subsequent PRF invocation.

One such common application arises in the context of deterministic signatures, which consists of generating a nonce by applying a PRF to the private key. In Schnorr and ECDSA, the nonce and a corresponding signature are sufficient to recover the private key. Thus, the nonce must also be distributed using the same secret-shared structure as the key. Distributed generation of deterministic signatures is once application that has both private input (the private key) and output (the nonce). Another example arises in the context of key derivation functions (KDFs), especially in a hierarchical structure, the output of the PRF may need to be used as an input (or even a key) for another evaluation of the PRF. A related application arises in the context of Bitcoin's BIP-32 derivation [64]. Motivated by such applications, we propose our LPN-wPRF candidate which has both its input and output over $\mathbb{Z}_2$.

Distributed FSS key generation. Function secret sharing (FSS) [18] is a useful tool for many cryptographic applications; see [22, 25] for recent examples. In many of these applications, two or more parties need to securely generate FSS keys, which in turn reduces to secure evaluation of a length-doubling PRG. Our LPN-style PRG candidate serves as a good basis for such protocols. In contrast to the black-box FSS key generation protocol of Doerner and shelat [38], its computational cost only scales logarithmically with the domain size. The optimal conjectured seed length of our PRG candidate ensures that FSS the key size is optimal as well.

1.1.5 Future directions.

Our work leaves several interesting avenues for further work. One direction is designing MPC protocols with malicious security while minimizing the extra cost. Recent techniques from [16, 26] can be helpful towards this goal. Another direction is designing and analyzing other symmetric primitives based on the alternating moduli paradigm. Relevant examples include hash functions, strong PRFs, and block ciphers. In fact, a strong PRF candidate was already suggested in [15]. Analyzing its concrete security is left for future work.

2 Preliminaries

Notation. We start with some basic notation. For a positive integer k, $[k]$ denotes the set $\{1, \ldots, k\}$. $\mathbb{Z}_p$ denotes the ring of integers modulo p. We use bold uppercase letters (e.g., $\mathbf{A}, \mathbf{K}$) to denote matrices. We use $\mathbf{0}^l$ and $\mathbf{1}^l$ to denote the all zeros and the all ones vector respectively (of length l), and drop l when sufficiently clear. For a vector x, by $x \bmod p$, we mean that each element in x is taken modulo p. We use $x \overset{\$}{\leftarrow} \mathcal{X}$ to denote sampling uniformly at random from domain $\mathcal{X}$. $\mathsf{Funcs}[\mathcal{X}, \mathcal{Y}]$ denotes the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$. $a \parallel b$ denotes concatenating the strings a and b.

For distributed protocols with N parties, we use $\mathcal{P} = \{\mathrm{P}_1, \ldots, \mathrm{P}_N\}$ to denote the set of parties. For a value x in group $\mathbb{G}$, we use $[\![x]\!]$ to denote an additive sharing of x (in $\mathbb{G}$) among the protocol parties, and $[\![x]\!]^{(i)}$ to denote the share of the ith party. When clear from context (e.g., a local protocol for P_i), we will often drop the superscript. When $\mathbb{G}' = \mathbb{G}^l$ is a product group (e.g., $\mathbb{Z}_p^l$), for $x \in \mathbb{G}'$, we may also say that $[\![x]\!]$ is a sharing *over* $\mathbb{G}$, similar to the standard practice of calling x a vector over $\mathbb{G}$.

For a value v in a group $\mathbb{G}$, we use $\tilde{v}$ to denote a random mask value sampled from the same group, and $\hat{v} = v + \tilde{v}$ (where $+$ is the group operation for $\mathbb{G}$) to denote v masked by $\tilde{v}$. We use the $+$ operator quite liberally and unless specified, it denotes the group operation (e.g., component-wise addition $\bmod\ p$ for $\mathbb{Z}_p^l$) for the summands.

Primitives. We briefly recall the standard definitions for the symmetric primitives we consider.

Definition 1 (Pseudorandom Function). *Let $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles of finite sets indexed by a security parameter λ. Consider an efficiently computable function family $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each function is given by $\mathsf{F}_\lambda : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. We say that $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is an (l, t, ε)-weak pseudorandom function if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda)$, the following holds: taking $f_\lambda \overset{\$}{\leftarrow} \mathsf{Funcs}[\mathcal{X}_\lambda, \mathcal{Y}_\lambda]$, $k \overset{\$}{\leftarrow} \mathcal{K}_\lambda$, and $x_1, \ldots, x_l \overset{\$}{\leftarrow} \mathcal{X}_\lambda$, we have that,*

$$\left|\Pr\left[\mathcal{A}\left(1^\lambda, \{x_i, \mathsf{F}_\lambda(k, x_i)\}_{i\in[l]}\right)\right] - \Pr\left[\mathcal{A}\left(1^\lambda, \{x_i, f_\lambda(x_i)\}_{i\in[l]}\right)\right]\right| \leq \varepsilon(\lambda).$$

Definition 2 (One-way Function). *Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles of finite sets indexed by a security parameter λ. Consider an efficiently computable function family $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each function is given by $\mathsf{F}_\lambda : \mathcal{X}_\lambda \to$*

$\mathcal{Y}_\lambda$. We say that $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is a (t,ε)-one-way function if for infinitely many $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$ running in time at most $t(\lambda)$, the following holds:

$$\Pr\left[x \xleftarrow{\$} \mathcal{X}; y \leftarrow \mathsf{F}_\lambda(x) : \mathsf{F}_\lambda(\mathcal{A}(1^{|x|}, y)) = y\right] \leq \varepsilon(\lambda)$$

Definition 3 (Pseudorandom Generator). *Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles of finite sets indexed by a security parameter λ. Consider an efficiently computable function family $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each function is given by $\mathsf{F}_\lambda : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. We say that $\{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is an (l,t,ε)-pseudorandom generator if F is length-expanding (i.e., $\forall\lambda, \forall x \in \mathcal{X}_\lambda$, $|x| < |\mathsf{F}_\lambda(x)|$) and for infinitely many $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$ running in time at most $t(\lambda)$, the following holds: taking $x_1,\dots,x_l \xleftarrow{\$} \mathcal{X}_\lambda$ $y_1,\dots,y_l \xleftarrow{\$} \mathcal{Y}_\lambda$, we have that,*

$$\left|\Pr\left[\mathcal{A}\left(1^\lambda, \{\mathsf{F}_\lambda(x_i)\}_{i\in[l]}\right)\right] - \Pr\left[\mathcal{A}\left(1^\lambda, \{y_i\}_{i\in[l]}\right)\right]\right| \leq \varepsilon(\lambda).$$

3 Candidate Constructions

In this section, we introduce our suite of candidate constructions for a number of cryptographic primitives: weak pseudo-random function families (wPRF), one-way functions (OWF), and pseudo-random generators (PRG). Our constructions are all based on similar interplays between mod-2 and mod-3 linear mappings.

3.1 Basic structure

Given the wide range of candidates we propose, we find it useful to have a clean way to describe the operations that are performed in our candidate constructions. For this, we take inspiration from the basic formalism of the function secret sharing (FSS) approach to MPC with preprocessing, first introduced by Boyle, Gilboa, and Ishai [20]. Abstractly, the technique starts by representing an MPC functionality as a circuit, where each gate represents an operation to be performed in the distributed protocol. Then the inputs and outputs of each gate are secret shared and the gate operation is "split" using an FSS scheme [18, 19]. To evaluate the circuit in a distributed fashion, the dealer first shares a random mask for each input wire in the circuit, and possibly additional correlated randomness. Now, to compute a gate, the masked input is first revealed to all parties, who can then locally compute shares of the output wire or shares of the masked output. This technique can also be viewed as a generalization of the TinyTable protocol [35].

While we find it useful to use the formalism from [20] for representing the circuit to be computed, we do not explicitly require the FSS formalism for splitting the functionality of each gate. The individual operations are quite straightforward, and we instead chose to directly provide the distributed protocols that compute them. Further, by doing so, our protocols can make better use of correlated randomness to reduce the overall protocol cost as compared to the general techniques in [20].

Circuit gates. We make use of just five types of basic operations, or "gates," which we detail below. All our constructions can be succinctly represented using just these gates. In Section 5, we will provide distributed protocols to compute them. Local protocols for the first two gates follow directly from the homomorphic properties of (additive) secret sharing but we mention them for completeness, as well as to enable a pictorial representation of constructions that use them. To cleanly describe both our candidate constructions and their distributed protocols, the gates we describe here depart from the formalism in [20] in that the input values of the circuit itself are not secret shared. The MPC protocols will instead take in secret shared inputs as necessary and evaluate the circuit in a distributed fashion. We denote by Gates, the set comprising of these gates.

- **Mod-p Public Linear Gate.** For a prime p, given a public matrix $\mathbf{A} \in \mathbb{Z}_p^{s\times l}$, the gate $\mathsf{Lin}_p^{\mathbf{A}}(\cdot)$ takes as input $x \in \mathbb{Z}_p^l$ and outputs $y = \mathbf{A}x \in \mathbb{Z}_p^s$.
- **Mod-p Addition Gate.** For a prime p, the gate $\mathsf{Add}_p(\cdot,\cdot)$ takes input $x, x' \in \mathbb{Z}_p^l$ and outputs $y = x + x' \bmod p$.
- **Mod-p Bilinear Gate.** For a prime p, and positive integers s and l, the gate $\mathsf{BL}_p^{s,l}(\cdot,\cdot)$ takes as input a matrix $\mathbf{K} \in \mathbb{Z}_p^{s\times l}$ and a vector $x \in \mathbb{Z}_p^l$ and outputs $y = \mathbf{K}x \in \mathbb{Z}_p^s$. When clear from context, we will drop the superscript and simply write $\mathsf{BL}_p(\mathbf{K}, x)$.
- **$\mathbb{Z}_2 \to \mathbb{Z}_3$ conversion.** For a positive integer l, the gate $\mathsf{Convert}_{(2,3)}^l(\cdot)$ takes as input a vector $x \in \mathbb{Z}_2^l$ and returns its equivalent representation x^* in $\mathbb{Z}_3^l$. When clear from context, we will drop the superscript and simply write $\mathsf{Convert}_{(2,3)}(x)$.
- **$\mathbb{Z}_3 \to \mathbb{Z}_2$ conversion.** For a positive integer l, the gate $\mathsf{Convert}_{(3,2)}^l(\cdot)$ takes as input a vector $x \in \mathbb{Z}_3^l$ and computes its mapping x^* in $\mathbb{Z}_2^l$. For this, each $\mathbb{Z}_3$ element in x is computed modulo 2 to get the corresponding $\mathbb{Z}_2$ element in the output x^*. Specifically, each 0 and 2 are mapped to 0 while each 1 is mapped to 1. When clear from context, we will drop the superscript and simply write $\mathsf{Convert}_{(3,2)}(x)$.

Note that the difference between the Lin and the BL gates is seen in the context of their distributed protocols. For Lin, the matrix $\mathbf{A}$ will be publicly available to all parties, while the input x will be secret shared. On the other hand, for BL, both the key $\mathbf{K}$ and the input x will be secret shared. We call this gate *bilinear* because its output is linear in both of its (secret-shared) inputs. Also note that although the $\mathsf{Convert}_{(2,3)}$ gate is effectively a no-op in a centralized evaluation, in the distributed setting, the gate will be used to convert an additive sharing over $\mathbb{Z}_2$ to an additive sharing over $\mathbb{Z}_3$.

As described previously, for the (bi)linear mappings, we focus only on constructions that use mod-2 and mod-3 mappings. In Fig. 1, we provide a pictorial representation for each circuit gate. We will connect these pieces together to also provide clean visual representations for all our constructions.

The homomorphic properties of additive secret sharing directly imply that the gates Lin and Add can be computed locally without any preprocessing or

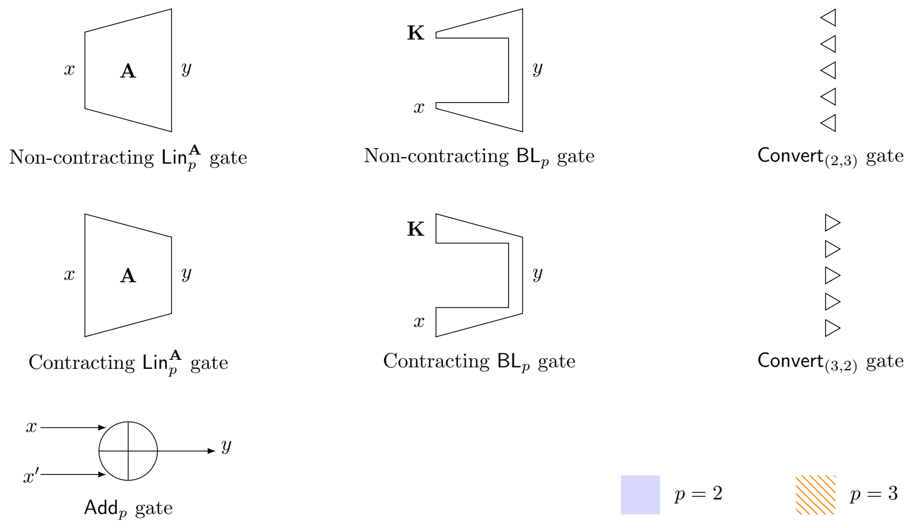

x
$\mathbf{A}$
y
Non-contracting $\mathsf{Lin}_p^{\mathbf{A}}$ gate
$\mathbf{K}$
x
y
Non-contracting BL_p gate
$\mathsf{Convert}_{(2,3)}$ gate
x
$\mathbf{A}$
y
Contracting $\mathsf{Lin}_p^{\mathbf{A}}$ gate
$\mathbf{K}$
x
y
Contracting BL_p gate
$\mathsf{Convert}_{(3,2)}$ gate
x
x'
y
Add_p gate
$p = 2$
$p = 3$

Fig. 1: Pictorial representations of the circuit gates. For the linear and bilinear gates, non-contracting means that the length of the output vector is greater than or equal to the length of the input vector, while contracting means that the output vector is smaller than the input vector. Additionally, for $p = 2$, the gates are shaded in violet, and for $p = 3$, the gates contain diagonal orange lines.

communication. For the other gates, we provide protocols to evaluate them in a distributed setting (i.e., where all inputs/outputs are secret shared) in Section 5.

Construction styles. The candidate constructions we introduce follow one of two broad styles which we detail below. A wPRF construction for the first style was first proposed by [15]. Here, we also propose a suite of symmetric primitives (e.g., OWFs, PRGs) with the same basic structure.

- (p, q)**-constructions.** For distinct primes p, q, the (p, q)-constructions have the following structure: On an input x over $\mathbb{Z}_p$, first a linear mod p mapping is applied, followed by a linear mod q mapping. Note that after the mod p mapping, the input is first reinterpreted as a vector over $\mathbb{Z}_q$. For unkeyed primitives (e.g., OWF), both mappings are public, while for keyed primitives (e.g., wPRF), the key is used for the first linear mapping. The construction is parameterized by positive integers n, m, t (functions of the security parameter λ) denoting the length of the input vector (over $\mathbb{Z}_p$), the length of the intermediate vector, and the length of the output vector (over $\mathbb{Z}_q$) respectively. The two linear mappings can be represented by matrices $\mathbf{A} \in \mathbb{Z}_p^{m \times n}$ and $\mathbf{B} \in \mathbb{Z}_q^{t \times m}$. For keyed primitives, the key $\mathbf{K} \in \mathbb{Z}_p^{m \times n}$ will be used instead of $\mathbf{A}$.

  Concretely, given an input $x \in \mathbb{Z}_p^n$, the construction output is of the form $y = \mathbf{B}w \in \mathbb{Z}_q^t$ where $w = \mathbf{A}x$ is first viewed over $\mathbb{Z}_q$. In this paper, we will

$(2,3)$-constructions

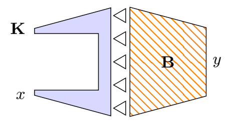
Parameters. Let λ be the security parameter and define parameters n, m, t as functions of λ such that $m \geq n, m \geq t$.
Public values. Let $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ and $\mathbf{B} \in \mathbb{Z}_3^{t \times m}$ be fixed public matrices chosen uniformly at random. The matrices can also be chosen to be full-rank circulant matrices.

Construction 1 (Mod-2/Mod-3 wPRF Candidate [15]) *The* $(2,3)$-wPRF *candidate is a family of functions* $\mathsf{F}_\lambda : \mathbb{Z}_2^{m \times n} \times \mathbb{Z}_2^n \to \mathbb{Z}_3^t$ *with key-space* $\mathcal{K}_\lambda = \mathbb{Z}_2^{m \times n}$, *input space* $\mathcal{X}_\lambda = \mathbb{Z}_2^n$ *and output space* $\mathcal{Y}_\lambda = \mathbb{Z}_3^t$. *For a key* $\mathbf{K} \in \mathcal{K}_\lambda$, *we define* $\mathsf{F}_{\mathbf{K}}(x) = \mathsf{F}_\lambda(\mathbf{K}, x)$ *as follows:*

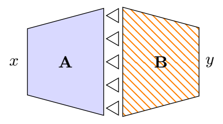
1. *On input* $x \in \mathbb{Z}_2^n$, *first compute* $w = \mathsf{BL}_2(\mathbf{K}, x) = \mathbf{K}x$.
2. *Output* $y = \mathsf{Lin}_3^{\mathbf{B}}\left(\mathsf{Convert}_{(2,3)}(w)\right)$. *That is, view* w *as a vector over* $\mathbb{Z}_3$ *and then output* $y = \mathbf{B}w$.

Construction 2 (Mod-2/Mod-3 OWF Candidate) *The* $(2,3)$-OWF *candidate is a function* $\mathsf{F}_\lambda : \mathbb{Z}_2^n \to \mathbb{Z}_3^t$ *with input space* $\mathcal{X}_\lambda = \mathbb{Z}_2^n$ *and output space* $\mathcal{Y}_\lambda = \mathbb{Z}_3^t$. *We define* $\mathsf{F}(x) = \mathsf{F}_\lambda(x)$ *as follows:*

1. *On input* $x \in \mathbb{Z}_2^n$, *first compute* $w = \mathsf{Lin}_2^{\mathbf{A}}(x) = \mathbf{A}x$.
2. *Output* $y = \mathsf{Lin}_3^{\mathbf{B}}\left(\mathsf{Convert}_{(2,3)}(w)\right)$. *That is, view* w *as a vector over* $\mathbb{Z}_3$ *and then output* $y = \mathbf{B}w$.



$(2,3)$-wPRF



$(2,3)$-OWF

Fig. 2: $(2,3)$-constructions

analyze this style of constructions for $(p,q) = (2,3)$ and $(3,2)$ since these are arguably the simplest constructions that employ linear mappings over alternate moduli. We find that the $(2,3)$-constructions outperform the $(3,2)$-constructions and we will primarily use the former style for our constructions.

- **LPN-style-constructions.** These constructions have the following general structure: On input x over $\mathbb{Z}_2$, first a linear mod 2 mapping given by the matrix $\mathbf{A}$ is applied to obtain u. Concurrently, the same linear mapping is also applied over $\mathbb{Z}_3$ (where both x and $\mathbf{A}$ are now reinterpreted over $\mathbb{Z}_3$) and then reduced modulo 2 to obtain v. The sum $w = u \oplus v$ is then multiplied by a second linear mapping (given by $\mathbf{B}$) over $\mathbb{Z}_2$. The mapping $\mathbf{B}$ is always public, while for keyed primitives, the key $\mathbf{K}$ is used instead of $\mathbf{A}$.

The construction is parameterized by positive integers n, m, t (as functions of the security parameter λ) denoting the size of the input vector, the intermediate vector(s), and the output vector (all over $\mathbb{Z}_p$). Concretely, given $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ and a public $\mathbf{B} \in \mathbb{Z}_2^{t \times m}$, for an input $x \in \mathbb{Z}_2^n$, the construction first computes the intermediate vector:

$$w = [(\mathbf{A}x \bmod 2) + (\mathbf{A}x \bmod 3) \bmod 2] \bmod 2.$$

LPN-style-constructions

Parameters. n, m, t are functions of the security parameter λ.
Public values. Let $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ and $\mathbf{B} \in \mathbb{Z}_2^{t\times m}$ be fixed public matrices chosen uniformly at random. Alternatively, the matrices can also be chosen to be full-rank circulant matrices.

Construction 3 (LPN-wPRF Candidate) *The* LPN-wPRF *candidate is a family of functions* $\mathsf{F}_\lambda : \mathbb{Z}_2^{m\times n} \times \mathbb{Z}_2^n \to \mathbb{Z}_2^t$ *with key-space* $\mathcal{K}_\lambda = \mathbb{Z}_2^{m\times n}$, *input space* $\mathcal{X}_\lambda = \mathbb{Z}_2^n$ *and output space* $\mathcal{Y}_\lambda = \mathbb{Z}_2^t$. *For a key* $\mathbf{K} \in \mathcal{K}_\lambda$, *we define* $\mathsf{F}_{\mathbf{K}}(x) = \mathsf{F}_\lambda(\mathbf{K}, x)$ *as follows:*

1. *On input* $x \in \mathbb{Z}_2^n$, *first compute* $u = \mathsf{BL}_2(\mathbf{K}, x) = \mathbf{K}x$.
2. *Let* $\mathbf{K}^* = \mathsf{Convert}_{(2,3)}(\mathbf{K})$ *and* $x^* = \mathsf{Convert}_{(2,3)}(x)$. *Compute* $v = \mathsf{Convert}_{(3,2)}(\mathsf{BL}_3(\mathbf{K}^*, x^*)) = \mathbf{K}^* x^* \bmod 2$. *That is, compute* $v = (\mathbf{K}x \bmod 3) \bmod 2$ *where both* $\mathbf{K}$ *and* x *are first reinterpreted over* $\mathbb{Z}_3$.
3. *Compute* $w = u \oplus v$ *and output* $y = \mathsf{Lin}_2^{\mathbf{B}}(w)$.

Construction 4 (LPN-PRG Candidate) *The* LPN-PRG *is a length-doubling PRG candidate defined as the function* $\mathsf{F}_\lambda : \mathbb{Z}_2^n \to \mathbb{Z}_2^{2n}$ *with input space* $\mathcal{X}_\lambda = \mathbb{Z}_2^n$ *and output space* $\mathcal{Y}_\lambda = \mathbb{Z}_2^{2n}$. *For this construction, we consider the parameters* n, m, t *with* $m \geq n, t$ *and* $t = 2n$. *We define* $\mathsf{F}(x) = \mathsf{F}_\lambda$ *as follows:*

1. *On input* $x \in \mathbb{Z}_2^n$, *first compute* $u = \mathsf{Lin}_2(\mathbf{A}, x) = \mathbf{A}x$.
2. *Let* $x^* = \mathsf{Convert}_{(2,3)}(x)$. *Compute* $v = \mathsf{Convert}_{(3,2)}(\mathsf{Lin}_3^{\mathbf{A}}(x^*)) = (\mathbf{A}x^*) \bmod 2$. *That is, compute* $(\mathbf{A}x \bmod 3) \bmod 2$ *where both* $\mathbf{A}$ *and* x *are first reinterpreted over* $\mathbb{Z}_3$.
3. *Compute* $w = u \oplus v$ *and output* $y = \mathsf{Lin}_2^{\mathbf{B}}(w)$.

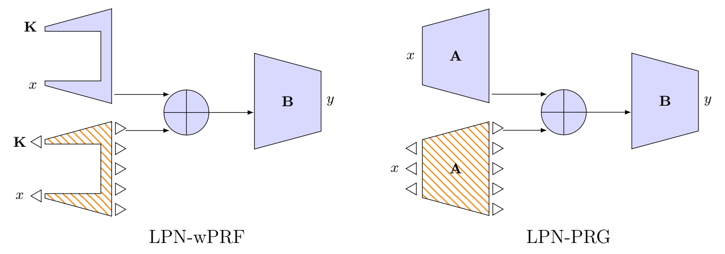


Fig. 3: LPN-style-constructions

The output y is then computed as $y = \mathbf{B}w \bmod 2$. The upshot of this style is that the input and the output are both over $\mathbb{Z}_2$. Intuitively, each intermediate vector bit can be thought of as a deterministic Learning-Parity-with-Noise (LPN) instance with a noise rate of 1/3. The noise is deterministically generated and is dependent on the input x and a specific column of $\mathbf{A}$. The noise for the i^{th} instance will be 1 if and only if $\langle \mathbf{A}_i, x \rangle = 1$.

A similar construction was considered in [15] but only for a single-bit output. Specifically, they considered $\mathbf{A} \in \mathbb{Z}_2^{1\times n}$ and output the single bit w. In our construction, we additionally apply a length-contracting linear mapping (using $\mathbf{B}$) to get the final output. This is done to resist standard attacks on LPN (see Section 4 and Appendix A for details).

Winning candidates. Through cryptanalysis and considering the cost for each candidate (See Sections 4 for details), we find that some of our candidates are more suited (i.e., "win") for a particular setting. Specifically, out of the candidates we consider, we find the following: $(2,3)$-wPRF and $(2,3)$-OWF are the best wPRF / OWF candidates with no restriction on the input/output space. LPN-wPRF is the best wPRF candidate when the input and output space are over $\mathbb{Z}_2$. LPN-PRG is the best PRG candidate. We provide formal and pictorial descriptions of our winning candidates in Fig. 2 and 3.

Structured keys. The constructions we described previously use general matrices in, e.g., $\mathbb{Z}_p^{m\times n}$. For keyed primitives, this results in a key size of mn elements of $\mathbb{Z}_p$ which is expensive to communicate within distributed protocols. Therefore, we will instead take advantage of structured matrices whose representation is only linear in n and m. Since both n and m are $O(\lambda)$ in our constructions, this reduces the communication complexity from quadratic to linear in λ. Furthermore, some structured matrices also benefit from asymptotically faster algorithms (e.g., FFT-based) for matrix multiplications and matrix-vector products. We briefly describe the types of structured matrices we utilize below. For this, consider a matrix $\mathbf{M} \in \mathbb{Z}_p^{m\times n}$.

- (Toeplitz matrices). A Toeplitz matrix, or a diagonal-constant matrix, is a matrix where each diagonal from left to right is constant. Specifically, $\mathbf{M}$ is Toeplitz if for all $i \in [m]; j \in [n]$, it holds that $M_{i,j} = M_{i+1,j+1}$ where $M_{i,j}$ denotes the element in row i and column j of $\mathbf{M}$. This means that a Toeplitz matrix can be represented by a single column and a single row, i.e., with $n + m - 1$ field elements.
- (Generalized circulant matrices). A generalized circulant matrix is a matrix where each row after the first, is a cyclic rotation of the first row. Specifically, if the first row of generalized circulant matrix $\mathbf{M}$ is the vector $(a_1, \ldots, a_n)$, then the m^{th} row of $\mathbf{M}$ will be given by the same vector cyclically rotated $m - 1$ times. In general, $m \neq n$, but the special case of $m = n$ (i.e., when the matrix is square) is called a (square) circulant matrix. Unless specified, for brevity, we will often use the term *circulant* to denote either generalized circulant matrices or the more specific (square) circulant matrices. This will not matter for our setting, since both can be efficiently represented using just n field elements.

We will usually instantiate our constructions using generalized circulant matrices to take advantage of their efficient representations. However, care must be taken while adding structure since this could potentially damage the security of a construction. Our cryptanalysis in Section 4 will therefore consider our constructions with structured matrices.

4 Cryptanalysis

We give a summary of cryptanalysis of our constructions, focusing on the main attacks that influence our parameters. The details are found in Appendix A.

4.1 Summary of Security Evaluation of the $(2, 3)$-OWF

The attacker is given $\hat{y} \in \mathbb{Z}_3^t$ and tries to invert it. Our most interesting attack on the $(2, 3)$-OWF is based on a reduction to subset-sum. We give the full details below.

Reduction to subset-sum. For a vector $w \in \mathbb{Z}_2^m$, there is an $(m - n) \times m$ (parity check) matrix $\mathbf{P}$ such that there exists $x \in \mathbb{Z}_2^n$ for which $\mathbf{A}x = w$ if and only if $\mathbf{P}w = 0$. Assume that $\hat{y}$ is the output of the $(2, 3)$-OWF on $x \in \mathbb{Z}_2^n$, and let $w = \mathbf{A}x$. Then, w satisfies the conditions $\mathbf{P}w = 0$ (over $\mathbb{Z}_2$) and $\mathbf{B}w = \hat{y}$ (over $\mathbb{Z}_3$). We attempt to find such w by a reduction to subset-sum, as detailed below. Suppose we find a set $J \subseteq [m]$ such that

$$\left(\sum_{j \in J} \mathbf{P}e_j \bmod 2, \sum_{j \in J} \mathbf{B}e_j \bmod 3 \right) = (\mathbf{0}, \hat{y}),$$

where $e_i \in \{0, 1\}^m$ is the i'th unit vector. Then, the preimage x can be computed by solving the linear equation system $\mathbf{A}x = \sum_{j \in J} e_j \bmod 2$.

Thus, we have reduced the problem to subset-sum with m binary variables $(\epsilon_1, \ldots, \epsilon_m) \in \{0, 1\}^m$, where we associate $\epsilon_i = 1$ with $(\mathbf{P}e_i, \mathbf{B}e_i) \in \mathbb{Z}_2^{m-n} \times \mathbb{Z}_3^t$, and define the target as $(\mathbf{0}, \hat{y}) \in \mathbb{Z}_2^{m-n} \times \mathbb{Z}_3^t$.

Solving the subset-sum problem. We can now apply the advanced subset-sum algorithm by Howgrave-Graham and Joux [46] and the more recent ones [9, 17], which are based on the *representation technique*. These algorithms were mostly designed to solve subset-sum problems over the integers. Below, we describe the main ideas of these algorithms and explain how to apply them to the special subset-sum problem we consider.

In the subset-sum problem over the integers, we are given a list of m positive integers $(a_1, a_2, \ldots, a_m)$ and another positive integer S such that $S = \sum_{i=1}^m \epsilon_i a_i$ for $\epsilon_i \in \{0, 1\}$. The goal is to recover the unknown coefficients ϵ_i.

A standard meet-in-the-middle approach for solving the problem has time complexity of about $2^{m/2}$. The representation technique gives an improved algorithm as briefly summarized below.

Assume that a solution to the subset-sum problem is chosen uniformly from $\{0, 1\}^m$ and the parameters are set such that the instance has about one solution on average. Effectively, this means that the density of the problem $d = \frac{n}{\log \max(\{a_i\}_{i=1}^m)}$ is set to 1.

The main idea of the basic algorithm of Howgrave-Graham and Joux [46] is to split the problem into two parts by writing $S = \sigma_1 + \sigma_2$, where $\sigma_1 = \sum_{i=1}^m \alpha_i a_i$, $\sigma_2 = \sum_{i=1}^m \beta_i a_i$ and $(\alpha_i, \beta_i) \in \{(0, 0), (0, 1), (1, 0)\}$. Thus, $\epsilon_i = \alpha_i + \beta_i$ for each i is a solution to the problem.

Note that each coefficient ϵ_i with value 1 can be split into $(0, 1)$, or $(1, 0)$. Thus, assuming that the solution has Hamming weight[4] of $m/2$ (which occurs

[4] In general, one may guess the Hamming weight of the solution and repeat the algorithm accordingly a polynomial number of times.

with probability $\Omega(1/\sqrt{m})$), it has $2^{m/2}$ different *representations*. Consequently, we may focus on finding only one of these representations by solving two subset-sum problems of Hamming weight $m/4$. Focusing on a single representation of the solution allows to beat the standard meet-in-the-middle approach which requires time $2^{m/2}$.

Adaptation of previous subset-sum algorithms. The algorithm of [46] can be easily adapted to our specialized subset-sum problem (although it is not defined over the integers). Moreover the improved algorithm of [9] considers additional representations of the solution by allowing α_i and β_i to also take the value -1 (implying that $\epsilon_i = 0$ can be decomposed into $(\alpha_i, \beta_i) \in \{(0,0), (-1,1), (1,-1)\}$). In our case, we associate $\alpha_i = -1$ with $(\mathbf{P}(-e_i), \mathbf{B}(-e_i)) = (\mathbf{P}e_i, 2 \cdot \mathbf{B}e_i) \in \mathbb{Z}_2^{m-n} \times \mathbb{Z}_3^t$. Finally, the recent improved algorithm of [17] considers representations over $\{-1,0,1,2\}$ and we can adapt this algorithm to our setting in a similar way.

In terms of complexity, ignoring polynomial factors in m, the attack of [46] runs in time $2^{0.337m}$ and uses $2^{0.256m}$ memory, while the complexity of attack of [17] requires $2^{0.283m}$ time and memory.

Thus, conservatively ignoring polynomial factors, for s-bit security we require $0.283m \geq s$, or $m \geq 3.53s$.

4.2 Summary of Security Evaluation of the $(2,3)$-wPRF

For the $(2,3)$-wPRF, the attacker obtains several samples $(x^{(1)}, \mathbf{B}, y^{(1)}), \ldots, (x^{(2^r)}, \mathbf{B}, y^{(2^r)})$ and tries to mount a key recovery and/or a distinguishing attack. We restrict the number of samples produced with a single secret to 2^{40}.

We will set the parameters such that $n - \log 3 \cdot t \geq s$, and thus there are 2^s keys on average that are consistent with a single sample. Therefore, any key recovery attack faster than 2^s has to make use of at least two samples. Particularly, the subset-sum attack can also be applied to the $(2,3)$-wPRF, but it is not clear how to use it efficiently on more than one sample (without strong relations between them).

The most important distinguishing attack looks for a bias in a linear combination of the output over $\mathbb{Z}_3$. Given a single sample $(x, \mathbf{B}, y)$, assume there exist $v \in \mathbb{Z}_3^m$ and $u \in \mathbb{Z}_3^t$ such that $u\mathbf{B} = v$ and the Hamming weight of v is ℓ. As $y = \mathbf{B}w \bmod 3$, the attacker computes $uy \bmod 3 = vw \bmod 3$ and thus obtains the value of a linear combination mod 3 of ℓ entries of $w \in \{0,1\}^m$. Since $w \in \mathbb{Z}_2^m$, this linear combination is biased, and the strength of the bias depends on how small ℓ is. The bias can be amplified using several samples. Consequently, we require that the rows of $\mathbf{B}$ do not span a vector of low Hamming weight. This analysis is probabilistic and leads to a lower bound on m.

Another important attack we consider exploits the fact that $\mathbf{K}$ is circulant and preserves symmetric properties of the input x (e.g., the two halves of x are equal). This attack imposes a lower bound on n so that such a symmetric vector in not found in the data, except with negligible probability. We leave it as an open problem to extend this basic attack.

Overall, we set $n = m = 2s$ and $t = s/\log 3$. These are somewhat aggressive parameters as the security margin against the above attacks in rather narrow. A choice of $n = m = 2.5s$ is more conservative.

4.3 Summary of Security Evaluation of the LPN-PRG

The attacker is given a single sample $\mathbf{A}, \mathbf{B}, y$ and tries to mount a key recovery and/or a distinguishing attack.

The construction differs from the alternative weak PRF construction proposed in [15] in two ways. The first transformation generates $t = 2n$ samples using a public matrix. Similarly to [15], each such sample can be viewed as an LPN sample, namely, a noisy linear equation over $\mathbb{Z}_2$ in the bits of the seed (although the noise is generated deterministically). However, in [15] $\mathbf{A}$ is a random matrix, whereas we use a (structured) Toeplitz matrix. This may be considered as weakening of the construction. On the other hand, the second transformation $\mathbf{B}$ "compresses" the samples and generally strengthens the construction.

A significant consideration in selecting the parameters is that the rows of $\mathbf{B}$ do not span a low Hamming weight vector, imposing a lower bound on m. Thus, only dense linear combinations of samples are available at the output, accumulating the noise. This should defeat standard attacks against LPN.

Overall, setting $n = s, m = 3s, t = 2s$ seems to provide sufficient resistance against the considered attacks.

4.4 Summary of Security Evaluation of the LPN-PRF

The attacks we consider against this primitive include a union of some of the attacks considered for the LPN-PRG and for the $(2, 3)$-wPRF constructions with some adjustments. Overall, we propose to set $n = m = 2s$ and $t = s$.

5 Distributed Protocols

We now describe efficient protocols to compute our candidate constructions in several interesting distributed settings. First, in Section 5.1, we provide a technical overview for our overall protocol design. Section 5.2 quantifies this approach by providing concrete costs for distributed evaluations for our $(2, 3)$-wPRF construction. We also provide two novel OPRF protocols based on this PRF in Section 5.3. In Appendix C, we also provide protocols for two more settings: the public-input setting (where the PRF input is known), and the 3PC setting without preprocessing where one party can be passively corrupt.

5.1 Technical Overview

Recall that all our constructions can be succinctly represented using five basic gates. The main strategy now will be to evaluate each of these gates in a

Protocol	Public Inputs	Shared Inputs	Shared Correlated Randomness	Output Shares (over base group $\mathbb{G}$)
$\pi_{\mathsf{Lin}}^{\mathbf{A},p}$	$\mathbf{A}$	x	-	$y = \mathbf{A}x$ (over $\mathbb{Z}_p$)
π_{Add}^{p}		x, x'	-	$y = x + x'$ (over $\mathbb{Z}_p$)
π_{BL}^{p}	$\hat{\mathbf{K}}, \hat{x}$	-	$\tilde{\mathbf{K}}, \tilde{x}, \tilde{\mathbf{K}}\tilde{x}$	$y = \mathbf{K}x$ (over $\mathbb{Z}_p$)
$\pi_{\mathsf{Convert}}^{(2,3)}$	$\hat{x}$ (over $\mathbb{Z}_2$)	-	$r = \tilde{x}$ (over $\mathbb{Z}_3$)	$x^* = x$ (over $\mathbb{Z}_3$)
$\pi_{\mathsf{Convert}}^{(3,2)}$	$\hat{x}$ (over $\mathbb{Z}_3$)	-	$u = \tilde{x} \bmod 2$ (over $\mathbb{Z}_2$) $v = (\tilde{x} + \mathbf{1} \bmod 3) \bmod 2$ (over $\mathbb{Z}_2$)	$x^* = x \bmod 2$ (over $\mathbb{Z}_2$)

Table 2: Summary of input, output, and randomness for circuit gate protocols.

distributed manner. These gate evaluation subprotocols can then be easily composed to evaluate the candidate constructions.

We begin with distributed protocols to evaluate each of the five gates. Abstractly, the goal of a gate protocol is to convert shares of the inputs to shares of the outputs (or shares of the masked output). To make our formalism cleaner, the gate protocols, by themselves, will involve no communication. Instead, they can additionally take in masked versions of the inputs, and possibly some additional correlated randomness. When composing gate protocols, whenever a masked input is needed, the parties will exchange their local shares to publicly reveal the masked value. This choice also prevents redoing the same communication when the masked value is already available from earlier gate evaluations.

5.1.1 Distributed Computation of Circuit Gates

We provide detailed (local) protocols to compute each circuit gate in this section. The description of inputs (including shared correlated randomness) and outputs for each gate protocol is also summarized in Table 2. Protocols for the Lin and Add gates directly follow from the homomorphic properties of additive secret sharing, while the protocol for the BL gate is essentially a generalization of Beaver's multiplication triples [8, 20]. In this section, we briefly provide our novel protocols for the $\mathsf{Convert}_{(2,3)}$ and $\mathsf{Convert}_{(3,2)}$ gates and defer the details (as well as other gate protocols) to Appendix C.1.

$\mathbb{Z}_2 \to \mathbb{Z}_3$ conversion protocol $\pi_{\mathsf{Convert}}^{(2,3)}$.

- **Functionality**: Abstractly, the goal of the $\mathbb{Z}_2 \to \mathbb{Z}_3$ conversion protocol is to convert a sharing of x over $\mathbb{Z}_2$ to a sharing of the same $x^* = x$, but now over $\mathbb{Z}_3$. For our purpose, the parties will be provided the masked input $\hat{x} = x \oplus \tilde{x}$ (i.e., masking is over $\mathbb{Z}_2$) directly along with correlated randomness that shares $\tilde{x}$ over $\mathbb{Z}_3$.
- **Preprocessing**: Each party is also provided with shares of the mask $r = \tilde{x}$ over $\mathbb{Z}_3$ as correlated randomness.
- **Protocol details**: For the protocol $\pi_{\mathsf{Convert}}^{(2,3)}(\hat{x} \mid r)$, each party proceeds as follows:
$$[\![x^*]\!]^{(i)} = [\![\hat{x}]\!]^{(i)} + [\![r]\!]^{(i)} + (\hat{x} \odot [\![r]\!]^{(i)}) \mod 3$$

Primitive	Construction	Param. (n, m, t)	Distributed 2PC (with preprocessing)		Distributed 3PC	Public-Input 2PC (with preprocessing)	
			Online Comm.	Prepr.	Online Comm.	Online Comm.	Prepr.
wPRF	(2, 3)-wPRF	$(256, 256, 81)$	$(1536, 4, 2)$	$(2348, 662)$	$(1430, 4, 1)$	$(512, 2, 1)$	$(1324, 406)$
	LPN-wPRF	$(256, 256, 128)$	$(2860, 6, 3)$	$(4995, 1730)$		$(1324, 4, 2)$	$(3160, 918)$
OWF	(2, 3)-OWF	$(128, 452, 81)$	$(904, 2, 1)$	$(2337, 717)$	$(2525, 4, 1)$	-	-
PRG	LPN-PRG	$(128, 512, 256)$	$(1880, 4, 2)$	$(4334, 1227)$		-	-

Table 3: Concrete MPC costs for our winning candidate constructions in three settings (Distributed 2PC (with preprocessing), 3PC, and Public-input 2PC) using our proposed parameters. For the distributed 2PC and the public-input 2PC settings, we provide the total online communication (bits, messages, rounds) and the preprocessing required in bits (without compression, with compression). For the compressed size of the preprocessing, we do not include values that can be reused (e.g., PRG seeds). For the distributed 3PC setting, we provide the total online communication cost (bits, messages, rounds) for our (2, 3)-constructions. The cost of the reusable PRG seeds is not included

where $\odot$ denotes the Hadamard (component-wise) product modulo 3.

$\mathbb{Z}_3 \to \mathbb{Z}_2$ conversion protocol $\pi^{(3,2)}_{\mathsf{Convert}}$.

- **Functionality**: Abstractly, the goal of the protocol is to convert a sharing of x over $\mathbb{Z}_3$ to a sharing of $x^* = x \bmod 2$ over $\mathbb{Z}_2$. For our purpose, the parties will be provided with the masked input $\hat{x} = x + \tilde{x} \bmod 3$ directly, along with correlated randomness over $\mathbb{Z}_3$ (see below).
- **Preprocessing**: Each party is also given shares (over $\mathbb{Z}_2$) of two vectors: $u = \tilde{x} \bmod 2$ and $v = (\tilde{x} + \mathbf{1} \bmod 3) \bmod 2$ as correlated randomness.
- **Protocol details**: For the protocol $\pi^{(3,2)}_{\mathsf{Convert}}(\hat{x} \mid u, v)$, each party computes its share of x^* as follows: For each position $j \in [l]$, $[\![x^*]\!]^{(i)}_j = 1 - [\![u]\!]^{(i)}_j - [\![v]\!]^{(i)}_j$,$[\![v]\!]^{(i)}_j$, $[\![u]\!]^{(i)}_j$ when $\hat{x}_j = 0, 1, 2$ respectively.

In Appendix C.2, we show a generic technique to evaluate any construction built using the previous five gates in a distributed fashion. We also analyze the communication and preprocessing costs. Abstractly, communication will only be needed before BL, $\mathsf{Convert}_{(2,3)}$, and $\mathsf{Convert}_{(3,2)}$ gates to reconstruct the masked input. In terms of preprocessing, if PRG seeds are used for compression, then the computation for the $\mathsf{BL}^{k,l}_p$, $\mathsf{Convert}^l_{(2,3)}$, and $\mathsf{Convert}^l_{(3,2)}$ gates will require a preprocessing of $\log_2 p \cdot k$ bits, $\log_2 3 \cdot l$ bits, and $2l$ bits respectively. Table 3 provides the concrete efficiency costs for our constructions in a variety of settings.

5.2 Distributed Evaluation

We briefly detail our 2-party protocol for (2, 3)-wPRF in the preprocessing model. In this setting, two parties, denoted by P_1 and P_2 hold shares of both the key $\mathbf{K}$ and the input x. The goal is to compute shares of the output y.

For this, we provide the parties with preprocessed tuples for the BL gate, and the $\mathsf{Convert}_{2,3}$ gate. To evaluate an input, the two parties first mask their shares of $\mathbf{K}$ and x, and exchange them to reveal $\hat{\mathbf{K}}$ and $\hat{x}$. Both parties use $\boldsymbol{\pi}_{\mathsf{BL}}$ to compute shares of the intermediate vector w. Then, they mask their shares and exchange them to reveal $\hat{w}$. The parties can now use the $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$ protocol followed by a local multiplication by $\mathbf{B}$ to obtain shares of the output y.

5.3 Oblivious Evaluation

While our distributed evaluation protocols from Section 5.2 can be used directly for semi-honest *oblivious* PRF, or OPRF, evaluation in the preprocessing model, here we provide two protocols in this setting whose efficiency rivals that of DDH-based OPRF protocols. Recall that in the OPRF setting, one party P_1 (called the "server") holds the key $\mathbf{K}$ and the other party P_2 (called the "client") holds the input x. The goal of the protocol is to have the client learn the output of the PRF for key $\mathbf{K}$ and input x, while the server learns nothing. We provide only a brief description of our protocols next, and defer the full details as well as a comparison to other protocols to Appendix C.4.

5.3.1 OPRF Protocol $\boldsymbol{\pi}^{\mathsf{oprf}}_1$

Our first OPRF protocol is in spirit similar to the distributed evaluation for the $(2,3)$-wPRF construction. Since $\mathbf{K}$ is known to the server, and x is known to the client, both parties do not need to exchange their shares to reconstruct the masked values $\hat{\mathbf{K}}$ and $\hat{x}$; the party that holds a value can mask it locally and send it to the other party. This allows us to decouple the server's message that masks its PRF key from the rest of the evaluation. To update the key, the server can simply send $\hat{\mathbf{K}} = \mathbf{K} + \tilde{\mathbf{K}}$ to the client. Many PRF evaluations can now be done using the same $\hat{\mathbf{K}}$. The upshot of this is that when the client already knows the key mask, the protocol has an optimal 2-round structure (one message from the client followed by one message from the server). For our parameters $(n = m = 256, t = 81)$, $\boldsymbol{\pi}^{\mathsf{oprf}}_1$ has 897 bits of total online communication for input evaluation. To update the key, the server sends a 256-bit message to the client.

5.3.2 OPRF Protocol $\boldsymbol{\pi}^{\mathsf{oprf}}_2$

For the second protocol, the server masks the PRF in a different way; a multiplicative mask is used instead of an additive one. This saves 256 bits in the online phase at the expense of a slower key update phase.

5.4 Generating the correlated randomness.

In this section we show how to generate the preprocessing we require efficiently and without a trusted dealer. We will focus on the 2-party setting specifically.

5.4.1 $(2,3)$-correlations from OT correlations

We provide a new technique to generate the correlations needed for the $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$ protocol. The key technique we use is to convert OT correlations to the types of correlations our protocols require. Since prior work [23, 70] has shown how to efficiently create OT-correlations, this implies that the correlations required for our protocols can also be efficiently generated. For a 1-out-of-2 OT correlation over $\mathbb{Z}_3$, P_1 holds (z_0, z_1) and P_2 holds (c, z_c) where $z_0, z_1 \overset{\$}{\leftarrow} \mathbb{Z}_3$, $c \in \mathbb{Z}_2$ and $z_c = z_0$ if $c = 0$ and $z_c = z_1$ if $c = 1$. We refer to $((z_0, z_1), (c, z_c))$ as an OT correlation pair.

Conversion technique. Recall that for the $\mathbb{Z}_2 \rightarrow \mathbb{Z}_3$ conversion protocol $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$, as preprocessing, a dealer provides the parties with shares of a bit-vector both over $\mathbb{Z}_2$ and $\mathbb{Z}_3$. For simplicity, we first consider the correlated randomness for a single element. To convert the sharing for a single bit, the dealer provides the following correlated randomness to the parties: P_1 is given (w_1, r_1) and P_2 is given (w_2, r_2) such that $w_1, w_2 \in \mathbb{Z}_2; r_1, r_2 \in \mathbb{Z}_3$ and $(w_1 + w_2) \bmod 2 = (r_1 + r_2) \bmod 3$. We refer to $((w_1, r_1), (w_2, r_2))$ as a $(2, 3)$-correlation pair.

We now show how to convert an OT-correlation into a $(2, 3)$-correlation. Suppose for now that we have the ability to "throw" away OT-correlations where $z_0 = z_1$. We will get rid of this assumption later by communicating a single message from P_1 to P_2 which will intuitively detail which OT correlations to discard.

Protocol 5 *Given a (1-out-of-2) OT correlation* $((z_0, z_1), (c, z_c))$ *over* $\mathbb{Z}_3$ *where* $z_0 \neq z_1$*, to generate a* $(2, 3)$*-correlation, the parties proceed as follows:*

- P_1 *computes*

$$(w_1, r_1) = \begin{cases} (0, z_0) & \text{if } z_1 = z_0 - 1 \bmod 3 \\ (1, z_1) & \text{if } z_0 = z_1 - 1 \bmod 3 \end{cases}$$

- P_2 *computes* $(w_2, r_2) = (c, -z_c \bmod 3)$.

This means that an OT correlation can locally be converted to a $(2, 3)$-correlation when $z_0 \neq z_1$. Since P_1 knows these values, it still needs to communicate to P_2 whether to use a given correlation or not. The communication can be compressed using the binary entropy function $\mathsf{H}_b(p)$ which computes the entropy of a Bernoulli process with probability p. This leads to a communication cost of $1.5l \cdot \mathsf{H}_b(1/3) \approx 1.377l$ for an l-length $(2, 3)$-correlation. We defer the details to Appendix C.5. As another upshot, this means that the required $(2, 3)$ correlations can be generated even during the first round of the online protocol.

5.4.2 $(3, 2)$-correlations from OT correlations

We now show how to convert OT-correlations to the correlations we require for the $\boldsymbol{\pi}^{(3,2)}_{\mathsf{Convert}}$ protocol. For this, we will need 1-out-of-3 OT correlations for 2-bit strings. Formally, in such a correlation, P_1 receives (z_0, z_1, z_2) where each z_j is a 2-bit string, while P_2 receives (c, z_c) where $c \in \mathbb{Z}_3$ and z_c is the corresponding z_j indexed by $j = c$. As before, these OT correlations can also be efficiently generated and compressed using existing work [23, 70].

Now, to convert a single $\mathbb{Z}_2$ element to $\mathbb{Z}_3$, our protocol requires the following correlated randomness: P_i is given $(\tilde{x}_i, u_i, v_i)$ where $\tilde{x}_i \in \mathbb{Z}_3$, $u_i, v_i \in \mathbb{Z}_2$ such that the following holds. Define $\tilde{x} = \tilde{x}_1 + \tilde{x} \bmod 3$, $u = u_1 + u_2 \bmod 2$, and $v = v_1 + v_2 \bmod 2$. Then, $u = \tilde{x} \bmod 2$ and $v = (\tilde{x} + 1 \bmod 3) \bmod 2$. We call this sharing between the two protocol parties a $(3, 2)$-correlation pair.

Protocol 6 *Given a (1-out-of-3) OT-correlation* $((z_0, z_1, z_2), (c, z_c))$ *for 2-bit strings, to generate a* $(3, 2)$*-correlation from this, the parties proceed as follows:*

- *First,* P_1 *samples its shares randomly as* $\tilde{x}_1 \overset{\$}{\leftarrow} \mathbb{Z}_3$, $u_1, v_1 \overset{\$}{\leftarrow} \mathbb{Z}_2$.
- *Now, for each* $j \in \mathbb{Z}_3$, P_1 *sets the 2-bit string* s_j *as follows. Let* $w = \tilde{x}_i + j \bmod 3$. *Then,* $s_j = (u_1 \parallel \neg v_1)$ *if* $w = 0$*;* $s_j = (\neg u_1 \parallel v_1)$ *if* $w = 1$*;* $s_j = (u_1 \parallel v_1)$ *if* $w = 2$. *Intuitively,* P_1 *sets the OT tuple to be what* P_2*'s share would be if it chose that particular index in an OT protocol.*
- P_1 *masks the* s_j *and sends them to* P_2. *Specifically,* P_1 *sends* $r_j \leftarrow s_j + z_j$ *(where each bit is added modulo 2) for each* $j \in \mathbb{Z}_3$.
- P_2 *sets* $\tilde{x}_2 \leftarrow c$*, and* $u_2 \parallel v_2 \leftarrow r_c$ *(i.e., the corresponding 2-bit string* r_c *sent by* P_1 *is parsed into* u_2 *and* v_2*)*
- *Finally, for the* $(3, 2)$*-correlation,* P_i *takes its share as* $(\tilde{x}_i, u_i, v_i)$

This is less efficient than our protocol to generate a $(2, 3)$ correlation and takes 6 bits of communication per instance. Note that the communication is still unidirectional as only P_1 sends a message. Consequently, the $(3, 2)$ correlations can also be generated on the fly given OT correlations as part of the first protocol round. More details are given in Appendix C.5.

6 Application: Signatures with the $(2, 3)$-OWF

Here we describe a signature scheme using the $(2, 3)$-OWF. Our presentation is tailored to the $(2, 3)$-OWF, but we note that this approach is general. All of the candidate primitives in this paper would be a suitable choice of F (note that they are all OWFs when the input is chosen at random) and we evaluated them all before settling on $(2, 3)$-OWF, which gives the shortest signatures.

Abstractly, a signature scheme can be built from any one-way function F that has an MPC protocol to evaluate it, by setting the public key to $y = \mathsf{F}(x)$ for a random secret x, and then proving knowledge of x, using a proof system based on the MPC-in-the-head paradigm [48]. In addition to assuming the OWF is secure, the only other assumption required is a secure hash function. As no additional number-theoretic assumptions are required, these types of signatures are often proposed as secure post-quantum schemes.

Concretely, our design follows the Picnic signature scheme [27], specifically the variant instantiated with the KKW proof system [54] (named Picnic2 and Picnic3). We chose to use the KKW, rather than ZKB++ proof system since our MPC protocol to evaluate the $(2, 3)$-OWF is most efficient with a pre-processing phase, and KKW generally produces shorter signatures. We replace the LowMC block cipher [2] in Picnic with the $(2, 3)$-OWF, and make the corresponding changes to the MPC protocol.

This is the first signature scheme based on the hardness of inverting the $(2,3)$-OWF (or similar function), a function with a simple mathematical description, making it an accessible target for cryptanalysis, especially when compared to block ciphers. Arguably, the simplicity of the OWF can lead to simpler implementations: the MPC protocol is simpler, and no large precomputed constants are required.

Our presentation is somewhat brief here, as many details are identical to Picnic, and the $(2,3)$-OWF MPC has been described in Section 5. Appendix D includes additional details.

Parameters. Let κ be a security parameter. The $(2,3)$-OWF parameters are denoted (n, m, t). The KKW parameters (N, M, τ) denote the number of parties N, the total number of MPC instances M, and the number τ of MPC instances where the verifier checks the online phase of simulation. The scheme also requires a cryptographic hash function.

Key generation. The signer chooses a random $x \in \mathbb{Z}_2^n$ as secret key, and a random seed $s \in \{0,1\}^\kappa$ such that s expands to matrices $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ and $\mathbf{B} \in \mathbb{Z}_3^{m \times t}$ that is full rank (using a suitable cryptographic function, such as the SHAKE extendable output function [55]). Compute $y = \mathsf{F}(x)$ and set (y, s) as the public key. Recall that the $(2,3)$-OWF is defined as $y = \mathsf{F}(x)$ where $x \in \mathbb{Z}_2^n$ and $y \in \mathbb{Z}_3^t$, and is computed as $y = \mathbf{B}(\mathbf{A}x)$ where $\mathbf{A}x$ is first cast to $\mathbb{Z}_3$.

MPC protocol. By combining the protocols for the gates $\boldsymbol{\pi}_{\mathsf{Add}}^{3}$, $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{A},2}$, $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{B},3}$, and $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ described in Section 5 we have an N-party protocol for the $(2,3)$-OWF. The most challenging and costly step (in terms of communication) is the conversion gate, all other operations are done locally by the parties. In Appendix D we describe this protocol in full detail.

Sign and verify. The signature generation and verification algorithms for the $(2,3)$-OWF signature scheme are given in Fig. 4 (of Appendix D). Here we give an overview. The prover simulates the preprocessing and online phase for all M MPC instances, and commits to the preprocessing values, and MPC inputs and outputs. Then she is challenged to open τ of the M MPC instances. The verifier will check the simulation of the online phase for these instances, by re-computing all values as the prover did for $N-1$ of the parties, and for remaining unopened party, the prover will provide the missing broadcast messages and commitments so that the verifier may complete the simulation and recompute all commitments. For the $M-\tau$ instances not chosen by the challenge, the verifier will check the preprocessing phase only, by recomputing the preprocessing phase as the prover did.

Parameter selection and signature size. In Appendix D we give the formula for estimating signatures sizes. The impact of OWF choice is limited to one term, which is the sum of the sizes of the MPC inputs, broadcast messages, and auxiliary values produced by preprocessing. Selecting the KKW parameters

(M, N, τ) once the MPC costs are known follows the approach in Picnic: a range of options are possible, and we try to select parameters that balance speed (number of MPC executions and number of parties) and size. Since the MPC costs of the $(2,3)$-OWF are very close to those of LowMC, the options follow a similar curve.

Table 6 gives some options with $N = 16, 64$ parties, providing 128 and 256 bits of security. For each category, we highlight the row of $(2,3)$-OWF parameters that are a direct comparison to Picnic. Signatures using the $(2,3)$-OWF are slightly shorter (five to fifteen percent) than Picnic using LowMC.

7 Implementation and Evaluation

We implemented our 2-party protocols to compute the $(2,3)$-wPRF candidate (Construction 3) both in the distributed setting (Section 5.2), and the oblivious evaluation setting (Sections 5.3.1 and 5.3.2). Our implementations are in C++. For the $(2,3)$-wPRF construction, we used the parameters $n = m = 256$ and $t = 81$. The implemented 23-constructions use a Toeplitz matrix in $\mathbb{Z}_2^{256\times 256}$ as the key, take as input a vector in $\mathbb{Z}_2^{256}$ and output a vector in $\mathbb{Z}_3^{81}$. The correlated randomness was implemented as if provided by a trusted third party. See Section 5.4 for concretely efficient protocols for securely generating the correlated randomness, which we did not implement but give efficiency estimates based on prior works.

Optimizations. We start with a centralized implementation of the 23-wPRF. We find optimizations that provide 25x better performance over a naïve implementation. We use three major optimizations in our implementation. First, we use *bit packing* for $\mathbb{Z}_2$ vectors through which we can pack several elements in a machine word and operate on them together in an SIMD manner. Second, we use *bit slicing* for $\mathbb{Z}_3$ vectors by representing them as a pair of $\mathbb{Z}_2$ vectors. All operations on the $\mathbb{Z}_3$ vectors can now be translated to operations on the $\mathbb{Z}_2$ vectors. Finally, we use a lookup table optimization for the final $\mathbb{Z}_3$ linear mapping (i.e., multiplication by $\mathbf{B}$). For this, we split the 256-column matrix $\mathbf{B}$ into 16 pieces with 16 columns each and store multiplications with all $\mathbb{Z}_3^{16}$ vectors for each piece. The size for each piece was decided as a tradeoff between the lookup table size and the computational efficiency. We defer the details for our optimization techniques to Appendix E and provide the benchmarks in Table 4.

7.1 Performance Benchmarks

Experimental setup. We ran all our experiments on a t2.medium AWS EC2 instance with 4GiB RAM (architecture: x86-64 Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz) running on Ubuntu 18.04. The performance benchmarks and timing results we provide are averaged over 1000 runs. For the distributed construction benchmarks, both parties were run on the same instance. We separately report the computational runtime for the parties, and analytically compute the communication costs.

Optimization			Runtime (μs)	Evaluations / sec
Packing	Bit Slicing	Lookup Table		
Baseline implementation			156.41	6K
✓			26.84	37K
✓	✓		18.5	65K
✓	✓	✓	6.08	165K

Table 4: Centralized 23-wPRF benchmarks for a baseline implementation and for different optimization techniques. Packing was done into 64-bit sized words (for both $\mathbb{Z}_2$ and $\mathbb{Z}_3$ vectors). For the lookup table optimization, a table with 81×2^{20} $\mathbb{Z}_3$ elements, or roughly of size 135MB, was preprocessed. Runtimes are all given in microseconds (μs).

Distributed wPRF evaluation. We implement our 2-party semi-honest distributed protocol for evaluating the $(2,3)$-wPRF construction and report timings for our implementation. Since this candidate was first proposed in [15], we also implement their protocol as a comparison point. For both protocols, we use the parameters $n = m = 256$, $t = 81$ for the PRF and use the same optimizations for an accurate comparison. We found that our protocol is better in all metrics. For a single evaluation, our protocol requires 12.12 μs, 662 bits of preprocessing, and 1536 bits of online communication. On the other hand, the protocol from [15] requires 28.02μs, 3533 bits of preprocessing, and 2612 bits of online communication for one evaluation.

OPRF evaluation. In Table 5, we provide performance benchmarks for both our oblivious protocols (see Section 5.3) for the $(2,3)$-wPRF construction. We also compare our results to the standard DDH-based OPRF (see Appendix C.4.3). For our timing results, we report both the server and client runtimes (averages over 1000 runs). For each construction, we also include the size of the preprocessed correlated randomness, and the online communication cost. All constructions are parameterized appropriately to provide 128-bit security.

For our constructions, we report separately, the timings for refreshing the key and evaluating the input. For the comparison with the DDH-based OPRF construction, we use the libsodium library [58] for the elliptic curve scalar multiplication operation. We use the Curve25519 elliptic curve, which has a 256-bit key size, and provides 128 bits of security.

References

[1] Adi Akavia et al. “Candidate weak pseudorandom functions in AC mod 2”. In: *ITCS 14*. 2014, pp. 251–260.

[2] Martin R. Albrecht et al. “Ciphers for MPC and FHE”. In: *EUROCRYPT*. 2015, pp. 430–454.

[3] Martin R. Albrecht et al. “Feistel Structures for MPC, and More”. In: *ESORICS*. 2019, pp. 151–171.

Protocol		Runtime (μs)		Preprocessing (bits)	Communication (bits)	
		Client	Server		Client	Server
π_1^{oprf}	Key Update	-	0.65	256	-	256
(Section 5.3.1)	Evaluation	8.54	9.45	2092	512	385
π_2^{oprf}	Key Update	-	3.16	256	-	256
(Section 5.3.2)	Evaluation	7.91	8.21	1836	256	385
DDH-based OPRF		57.38	28.69	-	256	256

Table 5: Comparison of protocols for (semi-honest) OPRF evaluation in the preprocessing model. Runtimes in microseconds (μs) are provided separately for refreshing the key (Key Update) and for evaluating an input (Evaluation). Communication and preprocessing are also provided separately for the two stages.

[4] Abdelrahaman Aly et al. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". In: *TOSC* 2020.3 (2020), pp. 1–45.

[5] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "Cryptography in NC^0". In: *FOCS*. 2004, pp. 166–175.

[6] Vladimir L'vovich Arlazarov et al. "On economical construction of the transitive closure of an oriented graph". In: *Doklady Akademii Nauk*. 1970, pp. 487–488.

[7] Carsten Baum et al. "Banquet: Short and Fast Signatures from AES". In: (2021). Available as IACR ePrint Report 2021/068., to appear.

[8] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *CRYPTO*. 1991, pp. 420–432.

[9] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. "Improved Generic Algorithms for Hard Knapsacks". In: *EUROCRYPT*. 2011, pp. 364–385.

[10] Mihir Bellare et al. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme". In: *J. Cryptology* 16 (3 2003), pp. 185–215.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *STOC*. 1988, pp. 1–10.

[12] Ward Beullens. "Sigma Protocols for MQ, PKP and SIS, and Fishy Signature Schemes". In: *EUROCRYPT*. 2020, pp. 183–211.

[13] Ward Beullens and Cyprien de Saint Guilhem. "LegRoast: Efficient Post-quantum Signatures from the Legendre PRF". In: *PQCRYPTO*. 2020, pp. 130–150.

[14] Manuel Blum and Silvio Micali. "How to Generate Cryptographically Strong Sequences of Pseudorandom Bits". In: *SICOMP* 13 (4), pp. 850–864.

[15] Dan Boneh et al. "Exploring Crypto Dark Matter: New Simple PRF Candidates and Their Applications". In: *TCC*. 2018, pp. 699–729.

[16] Dan Boneh et al. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: *CRYPTO*. Ed. by Alexandra Boldyreva and Daniele Micciancio. 2019, pp. 67–97.

[17] Xavier Bonnetain et al. "Improved Classical and Quantum Algorithms for Subset-Sum". In: *ASIACRYPT*. 2020, pp. 633–666.

[18] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *EUROCRYPT*. 2015, pp. 337–367.

[19] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing: Improvements and Extensions". In: *CCS*. 2016, pp. 1292–1303.

[20] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Secure Computation with Preprocessing via Function Secret Sharing". In: *TCC*. 2019, pp. 341–371.

[21] Elette Boyle et al. "Compressing Vector OLE". In: *CCS*. 2018, pp. 896–912.

[22] Elette Boyle et al. "Efficient Pseudorandom Correlation Generators from Ring-LPN". In: *CRYPTO*. 2020, pp. 387–416.

[23] Elette Boyle et al. "Efficient Pseudorandom Correlation Generators: Silent OT Extension and More". In: *CRYPTO*. 2019, pp. 489–518.

[24] Elette Boyle et al. "Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation". In: *CCS*. 2019, pp. 291–308.

[25] Elette Boyle et al. "Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation". In: *Cryptology ePrint Archive, Report 2020/1392* (2020).

[26] Elette Boyle et al. "Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs". In: *CCS*. 2019, pp. 869–886.

[27] Melissa Chase et al. "Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives". In: *CCS*. 2017, pp. 1825–1842.

[28] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty Unconditionally Secure Protocols". In: *STOC*. 1988, pp. 11–19.

[29] Lijie Chen. "Non-deterministic Quasi-Polynomial Time is Average-Case Hard for ACC Circuits". In: *FOCS*. 2019, pp. 1281–1304.

[30] Lijie Chen and Hanlin Ren. "Strong average-case lower bounds from non-trivial derandomization". In: *STOC*. 2020, pp. 1327–1334.

[31] Jung Hee Cheon et al. "Adventures in Crypto Dark Matter: Attacks, Fixes and Analysis for Weak Pseudorandom Function Candidates". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 783.

[32] Jung Hee Cheon et al. *Adventures in Crypto Dark Matter: Attacks, Fixes for Weak Pseudorandom Function Candidates.* Cryptology ePrint Archive, Report 2020/783. 2020.

[33] Geoffroy Couteau et al. "On the Concrete Security of Goldreich's Pseudorandom Generator". In: *ASIACRYPT*. 2018, pp. 96–124.

[34] Ivan Damgård and Marcel Keller. "Secure Multiparty AES". In: *FC*. 2010, pp. 367–374.

[35] Ivan Damgård et al. "The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited". In: *EUROCRYPT*. 2017, pp. 167–187.

[36] Ivan Bjerre Damgård. "On The Randomness of Legendre and Jacobi Sequences". In: *CRYPTO*. 1988, pp. 161–172.

[37] Itai Dinur and Niv Nadler. “Multi-target Attacks on the Picnic Signature Scheme and Related Protocols”. In: *EUROCRYPT*. 2019, pp. 699–727.

[38] Jack Doerner and Abhi Shelat. “Scaling ORAM for Secure Computation”. In: *CCS*. Ed. by Bhavani M. Thuraisingham et al. 2017, pp. 523–535.

[39] Andre Esser, Robert Kübler, and Alexander May. “LPN Decoded”. In: *CRYPTO*. 2017, pp. 486–514.

[40] Yuval Filmus et al. “Limits of Preprocessing”. In: *CCC*. 2020, 17:1–17:22.

[41] Michael J. Freedman et al. “Keyword Search and Oblivious Pseudorandom Functions”. In: *TCC*. 2005, pp. 303–324.

[42] Oded Goldreich. “Candidate One-Way Functions Based on Expander Graphs”. In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation.* Vol. 6650. Lecture Notes in Computer Science. 2011, pp. 76–87.

[43] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “On the cryptographic applications of random functions”. In: *CRYPTO*. 1984, pp. 276–288.

[44] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *STOC*. 1987, pp. 218–229.

[45] Lorenzo Grassi et al. “MPC-Friendly Symmetric Key Primitives”. In: *CCS*. 2016, 430–443.

[46] Nick Howgrave-Graham and Antoine Joux. “New Generic Algorithms for Hard Knapsacks”. In: *EUROCRYPT*. 2010, pp. 235–256.

[47] Yuval Ishai et al. “Extending Oblivious Transfers Efficiently”. In: *CRYPTO*. Ed. by Dan Boneh. 2003, pp. 145–161.

[48] Yuval Ishai et al. “Zero-knowledge from secure multiparty computation”. In: *STOC*. 2007, pp. 21–30.

[49] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. “Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model”. In: *ASIACRYPT*. 2014, pp. 233–253.

[50] Stanislaw Jarecki and Xiaomin Liu. “Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection”. In: *TCC*. 2009, pp. 577–594.

[51] Stanislaw Jarecki et al. “Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online)”. In: *EURO S&P*. 2016, pp. 276–291.

[52] Valentine Kabanets et al. “Algorithms and Lower Bounds for De Morgan Formulas of Low-Communication Leaf Gates”. In: *CCC*. 2020, 15:1–15:41.

[53] Daniel Kales and Greg Zaverucha. “Improving the Performance of the Picnic Signature Scheme”. In: *TCHES* 2020 (4 2020), pp. 154–188.

[54] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. “Improved Non- Interactive Zero Knowledge with Applications to Post-Quantum Signatures”. In: *CCS*. 2018, pp. 525–537.

[55] J. Kelsey, S. J. Chang, and R. Perlner. *SHA-3 derived functions: cSHAKE KMAC TupleHash and ParallelHash.* National Institute for Standards and Technology, Special Publication 800-185. 2016.

[56] Vladimir Kolesnikov et al. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *CCS*. 2016, pp. 818–829.

[57] Leonid Levin. "One-Way Functions and Pseudorandom Generators". In: *STOC*. 1985, pp. 363–365.

[58] *libsodium 1.0.18-stable*. https://libsodium.gitbook.io/doc/. Online; December 31 2020. 2020.

[59] Tsutomu Matsumoto and Hideki Imai. "Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption". In: *EUROCRYPT '88*. 1988, pp. 419–453.

[60] Daniele Micciancio and Michael Walter. "On the Bit Security of Cryptographic Primitives". In: *EUROCRYPT*. 2018, pp. 3–28.

[61] Eric Miles and Emanuele Viola. "Substitution-Permutation Networks, Pseudorandom Functions, and Natural Proofs". In: *J. ACM* 62.6 (2015), 46:1–46:29.

[62] Moni Naor, Benny Pinkas, and Omer Reingold. "Distributed pseudo-random functions and KDCs." In: *EUROCRYPT*. 1999, pp. 327–346.

[63] Benny Pinkas et al. "Secure Two-Party Computation Is Practical". In: *ASIACRYPT*. 2009, pp. 250–267.

[64] Bitcoin Improvement Proposal. *Hierarchical Deterministic Wallets*. https://en.bitcoin.it/wiki/BIP_0032. 2017.

[65] Cyprien Delpech de Saint Guilhem et al. "BBQ: Using AES in Picnic Signatures". In: *SAC*. 2019, pp. 669–692.

[66] Phillipp Schoppmann et al. "Distributed Vector-OLE: Improved Constructions and Implementation". In: *CCS*. 2019, pp. 1055–1072.

[67] István András Seres, Máté Horváth, and Péter Burcsi. *The Legendre Pseudorandom Function as a Multivariate Quadratic Cryptosystem: Security and Applications*. Cryptology ePrint Archive, Report 2021/182. 2021.

[68] The Picnic Design Team. *The Picnic Signature Algorithm Specification*. Version 3.0, Available at https://microsoft.github.io/Picnic/. 2020.

[69] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. "Global-Scale Secure Multiparty Computation". In: *CCS*. 2017, pp. 39–56.

[70] Kang Yang et al. "Ferret: Fast Extension for Correlated OT with Small Communication". In: *CCS*. 2020, pp. 1607–1626.

[71] Andrew C. Yao. "Theory and application of trapdoor functions". In: *FOCS*. 1982, pp. 80–91.

[72] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets". In: *FOCS*. 1986, pp. 162–167.

SUPPLEMENTARY MATERIAL

A Detailed Cryptanalysis

In this appendix we describe in detail the cryptanalysis of our new constructions. We start with some general remarks and then analyze each construction.

A.0.1 Choosing public inputs.

All the cryptosystems we define receive public inputs chosen at random. For example, the $(2, 3)$-OWF receives matrices $\mathbf{A}$ and $\mathbf{B}$ as public inputs. One option is to choose $\mathbf{A}$ and $\mathbf{B}$ independently per secret input. While an alternative option is to fix one (or even both) of the matrices and reuse them. Generally, this alternative option is more susceptible to multi-target attacks and attacks that are based on self-similarity. Thus, in general, the first option is considered more secure and this is the option we use. For similar reasons we choose the public parameters independently per secret for the PRFs and the PRG we define.

Of course, there are bad choices of the public inputs which could degrade security, and we need to show that these are unlikely to be occur.

Finally, for the PRFs we define, this still leaves open the question of how to select the public inputs x and B per sample computed with a secret key. We chose to select x independently per sample, while fixing $\mathbf{B}$ per secret key, as this allows to optimize performance by preprocessing. In terms of security, the choice of fixing $\mathbf{B}$ does allow for a wider range of attacks that we need to consider, as we demonstrate in the security analysis.

Restricted public matrices. In order to optimize performance, we select the public matrices for the schemes as random Toeplitz matrices. These matrices define a pairwise independent hash family and are known to satisfy the Gilbert–Varshamov bound for linear codes, which is the main tool we use in the analysis.

Private matrices for PRF constructions. The private matrix $\mathbf{K}$ in the PRF constructions is a circulant matrix defined by rotations of a secret $k \in \{0, 1\}^n$. While a choice of K with a small rank deficiency does not seem to have a significant impact on the security, some attacks on the schemes (particularly on the 2-3 PRF) may exploit matrices of particularly low rank (as $w = \mathbf{K}x \bmod 2$ resides in a subspace of small dimension).

Thus, if possible, it is preferable to select k such that $\mathbf{K}$ is a full rank matrix. Yet, this may require additional communication is distributed protocols. Otherwise, we need to understand the rank distribution of $\mathbf{K}$ when k is selected uniformly at random and prove that $\mathbf{K}$ has very small rank with negligible probability. For particular choices of n the analysis is simple.

Proposition 1. *Let $n = 2^{n'}$ for a positive integer n' and let $\mathbf{K} \in \mathbb{Z}_2^{n \times n}$ be a circulant matrix selected uniformly at random. Then, for any $a \in \{0, \dots n\}$, $\Pr[rank(K) \leq a] = 2^{-n+a}$.*

Proof. For every vector $u \in \mathbb{Z}_2^n$ we associate a polynomial of degree at most $n-1$, $u(x) = \sum_{j=0}^{n-1} u_j x^j \in \mathbb{Z}_2[x]$.

Assume that $\mathbf{K}$ is formed by rotations of $k \in \mathbb{Z}_2^n$. Then for $i \in \{0, \dots, n-1\}$, the i'th column of K is associated with the polynomial $x^i \cdot k(x) \bmod x^n - 1$. Thus, for $u \in \mathbb{Z}_2^n$, $\mathbf{K}u$ is given by the coefficients of the polynomial $u(x) \cdot k(x) \bmod x^n - 1$, and $Ku = 0$ if and only if $u(x) \cdot k(x)$ divides $x^n - 1$.

Therefore, there is a bijection between the kernel space of $\mathbf{K}$ of the subspace of polynomials $u(x) \in \mathbb{Z}_2[x]$ of degree at most $n-1$ that satisfy $u(x) \cdot k(x) \bmod x^n - 1 = 0$. The dimension of this subspace of polynomials is equal to the degree of $\gcd(k(x), x^n - 1)$ over $\mathbb{Z}_2[x]$.

Recalling that $n = 2^{n'}$, over $\mathbb{Z}_2[x]$ we have $x^{2^{n'}} - 1 = (x-1)^{2^{n'}} = (x-1)^n$. To conclude the proof, $rank(\mathbf{K}) \leq a$ if and only if the kernel dimension of $\mathbf{K}$ is at least $n - a$. Equivalently, the degree of $\gcd(k(x), (x-1)^n)$ is at least $n - a$, or $k(x)$ divides $(x-1)^{n-a}$. There are exactly 2^a polynomials of degree at most $n - 1$ that divide $(x-1)^{n-a}$ and the probability of selecting one of them is $\frac{2^a}{2^n} = 2^{-n+a}$ as claimed.

A.0.2 Concrete security goals.

For each scheme the goal is to obtain parameter sets n, m, t such that it offers s-bit security, as defined below, while achieving good performance in distributed protocols.

OWF security. Given a OWF scheme $F(\cdot)$ (applied to a secret input, where the public parameters are embedded into F), we define an inversion attack game by choosing $\hat{x} \in \mathbb{Z}_2^n$ uniformly at random and giving $\hat{y} = F(\hat{x})$ to the adversary, whose goal to output some $x \in \mathbb{Z}_2^n$ such that $F(x) = \hat{y}$. We say that $F(\cdot)$ has s bits of security if no adversary can win the inversion attack game on $F(\cdot)$ with average complexity below 2^{s-1}.

Weak PRF security. We refer to both weak PRFs we define.

The key $k \in \mathbb{Z}_2^n$ that defines the secret matrix $\mathbf{K}$ is chosen uniformly at random. Moreover, the public matrix $\mathbf{B} \in \mathbb{Z}_3^{t \times m}$ (or $\mathbf{B} \in \mathbb{Z}_2^{t \times m}$) is chosen uniformly at random (or as a random Toeplitz matrix). For a parameter r, an adversary is given 2^r samples $(x^{(1)}, \mathbf{B}, y^{(1)}), \dots, (x^{(2^r)}, \mathbf{B}, y^{(2^r)})$, where each $x^{(i)} \in \mathbb{Z}_2^n$ is chosen independently and uniformly at random.

We will place a restriction of $r \leq 40$, corresponding to a practical limit of 2^{40} on the number of samples available to the adversary.

We consider two types of adversaries. The first type is a distinguisher that attempts to distinguish 2^r samples where each $y^{(i)}$ is generated using the PRF with a fixed k from 2^r samples where each vector $y^{(i)}$ is chosen uniformly at random. The second type attempts to find the secret key given samples generated using the PRF.

We say that the PRF has s bits of security if given (at most) 2^r samples both conditions below hold.

1. The advantage of a distinguishing adversary that runs in time 2^{τ} is at most $2^{(\tau-s)/2}$.
2. The probability that an adversary that runs in time 2^{τ} find the key is at most $2^{\tau-s}$.

This definition is aligned with the work of Micciancio and Walter [60].

PRG security. The secret seed $x \in \mathbb{Z}_2^n$ is chosen uniformly at random, along with the public parameters $\mathbf{A}, \mathbf{B}$. The adversary is given a single sample $\mathbf{A}, \mathbf{B}, y$. As for PRFs, security is defined against the two types of adversaries and the definition is similar.

A.0.3 Algebraic attacks.

In algebraic attacks the attacker represents the outputs (on internal variables) of the cipher as multivariate polynomials in the secret key (or preimage), obtaining a system of polynomial equations. The attacker then attempts to the solve the system using techniques such as linearization or applying algorithms for finding a reduced representation of the ideal generated by the polynomials in the form of a Gröbner basis. Such methods are known to be efficient only in particular cases where the polynomials have a special structure, or the polynomials equations are of low degree and the attacker obtains sufficiently many equations to solve the system by linearization.

In our case, the output of the schemes we define mix between the sums mod 2 and mod 3. For example, in the $(2,3)$-OWF and PRF constructions each output entry is a sum mod 3 of entries of w, where each such entry is a sum mod 2 of the unknown bits of the secret input. Due to the mix between the sums mod 2 and mod 3 we conjecture (similarly to [15]) that the output cannot be represented (or well-approximated) by a low degree polynomial over any specific polynomial ring. In particular, it was shown in [15] that the sum mod 3 of ℓ binary-valued variables is a high-degree polynomial over $\mathbb{Z}_2$, as long as ℓ is large (e.g., $\ell \approx n$). Crucially, our choice of parameters will ensure that the sums mod 2 and mod 3 are dense and contain many terms. In particular, for the $(2,3)$-OWF and PRF constructions we will make sure that the linear code spanned by the rows of B has large minimal distance, except with very small probability. Overall, we do not expect algebraic attacks to pose a threat to our schemes, and our analysis is mainly based on combinatorial attacks that attempt to recover the secret, or on statistical attacks whose goal is to distinguish the output from random.

A.1 Security Evaluation of the $(2,3)$-OWF

In this section we analyze the security of the $(2,3)$-OWF. Beforehand, we note that we may assume without loss of generality that in the most efficient construction, the number of expected preimages is (about) 1. Specifically, in our case, we may assume that $n = \log 3 \cdot t$ (up to rounding factors).

Indeed, setting $\log 3 \cdot t > n$ does not reduce the average number of preimages substantially. Consequently, any attack on a scheme with $n = \log 3 \cdot t'$ can be applied to a scheme with $\log 3 \cdot t > n$ by truncating the output to be of length $\log 3 \cdot t'$. Hence a scheme in which $\log 3 \cdot t > n$ does not offer better security than the truncated one. On the other hand, the truncated scheme has shorter output and is generally more efficient. Similarly, if $n > \log 3 \cdot t$, an attacker can fix $n - \log 3 \cdot t$ bits of the secret input to an arbitrary value and try to invert the image of the induced scheme where $n' = \log 3 \cdot t$ (note that on average, such a preimage exists).

A.1.1 Basic attacks.

We describe several basic attacks and analyze their complexity as a function of n, m, t. First, by exhaustive search, we can invert the (2,3)-OWF in time complexity 2^n or $3^t = 2^{\log 3 \cdot t}$.

Focusing on the value of m, by exhaustive search, we can find x such that $\mathbf{A}x = \mathbf{A}\hat{x}$ (which implies that the outputs are identical) in time complexity 2^m. A tighter restriction on m is imposed by the following attack: guess $m - t$ bits of $w = \mathbf{A}x$ and solve the linear equation system $\hat{y} = \mathbf{B}w$ over $\mathbb{Z}_3$ (which has t equations and variables) to obtain a full suggestion for w. A suggestion for w allows to compute x by solving the linear equation system $\mathbf{A}x = w$ over $\mathbb{Z}_2$. This attack has complexity 2^{m-t}. An improved attack is described next.

Enumerating $\boldsymbol{w}$ values. We show how to enumerate over all $w \in \{0,1\}^m$ that satisfy $\mathbf{B}w = \hat{y}$ in time complexity of about $2^{m/2}$ if $m \leq 2\log 3 \cdot t = 2n$, and $2^{m - \log 3 \cdot t} = 2^{m-n}$, otherwise.

Given such an algorithm, we can test each w by solving the equation system $\mathbf{A}x = w$ over $\mathbb{Z}_2$, and if a solution exists, we have successfully inverted $\hat{y}$.

Observe that if w and w' do not have a common 1 entry, then $w + w' \bmod 2 = w + w' \bmod 3$ (where the addition is performed entry-wise). Therefore,

$$\begin{aligned} \mathbf{B}(w + w' \bmod 2) \bmod 3 = \\ \mathbf{B}(w + w' \bmod 3) \bmod 3 = \\ (\mathbf{B}w \bmod 3) + (\mathbf{B}w' \bmod 3) \bmod 3. \end{aligned} \tag{1}$$

We use this observation in the following algorithm, whose complexity as claimed above.

1. Partition the m indices of w into 2 subsets I_1 and $I_2 = [m] \backslash I_1$, each of size $m/2$ bits.
2. For $i \in \{0, 1, \ldots 2^{m/2} - 1\}$, let w_i be the m-bit vector whose value on the $m/2$ indices of I_1 is i, and is 0 on the indices of I_2. For each such i, evaluate $\mathbf{B}w_i \bmod 3 = y_i$ and store the pairs (w_i, y_i) in a table $\mathcal{T}$, sorted by y_i values.
3. For $j \in \{0, 1, \ldots 2^{m/2} - 1\}$, let w'_j be the m-bit vector whose value on the $m/2$ indices of I_2 is j, and is 0 on the indices of I_1. For each such j, evaluate $\mathbf{B}w'_j \bmod 3 = y'_j$ and search $\mathcal{T}$ for the value $\hat{y} - y'_j \bmod 3$. If there exists a match y_i such that $y_i = \hat{y} - y'_j \bmod 3$ (or $y_i + y'_j \bmod 3 = \hat{y}$), recover the value w_i such that $\mathbf{B}w_i \bmod 3 = y_i$ from $\mathcal{T}$ and return $w = w_i + w'_j \bmod 2$.

Note that the expected number of $w \in \{0,1\}^m$ that satisfy $\mathbf{B}w = \hat{y}$ is $2^{m-\log 3 \cdot t}$. Hence, we cannot hope to obtain better complexity than $2^{m-\log 3 \cdot t}$ without exploiting additional constraints on w, imposed by the matrix $\mathbf{A}$. Our subset-sum reduction (given in Section 4) shows how this can be done.

Induced schemes. Given the scheme $(2,3)$-OWF with a matrix $\mathbf{B}$ and output y such that $\mathbf{B}w = y$, and any positive integer r, we can left-multiply both sides by any $r \times t$ matrix $\mathbf{C}$ over $\mathbb{Z}_3$ to obtain $\mathbf{CB}w = \mathbf{C}y$. Note that each row of the matrix $\mathbf{CB}$ is a linear combination of the rows of $\mathbf{B}$. Using such a matrix $\mathbf{C}$, we can perform Gaussian elimination on the rows of $\mathbf{B}$.

We denote the resultant induced scheme by $OWF_{\mathbf{C}}(\cdot)$. Observe that if $OWF(x) = y$, then $OWF_{\mathbf{C}}(x) = \mathbf{C}y$. We now describe a simple attack that uses an induced scheme where $\mathbf{C}$ is only a row vector.

Low Hamming weight combinations of the rows of B. Assume that there is a vector $v \in \mathbb{Z}_3^m$ of Hamming weight ℓ in the row space of $\mathbf{B}$, namely, there exists a vector $u \in \mathbb{Z}_3^t$ for which $u\mathbf{B} = v$. If ℓ is sufficiently small, then we could use the induced scheme $OWF_u(\cdot)$ to speed up exhaustive search.

Denote the set of ℓ non-zero indices of v by I. Given $\hat{y} = \mathbf{B}w \bmod 3$, we compute the value of $u\hat{y} \bmod 3 = vw \bmod 3$. We can now enumerate the values of the corresponding set I of ℓ bits of w for which $u\hat{y} \bmod 3 = vw \bmod 3$ holds. This set of bits has $\frac{2^\ell}{3}$ possible values. Each such ℓ-bit value gives rise of a system of ℓ linear equations on x, and we exhaustively search its solution space of size $2^{n-\ell}$. Overall, if $\ell \leq n$ the complexity of the attack is $\frac{2^\ell}{3}$, while if $\ell = n+1$, the complexity is $\frac{2^{\ell+1}}{3}$. When $\ell > n+1$, the complexity is higher than 2^n. Thus, we will require that such a vector v of low Hamming weight about n does not exist, except with small probability. This probability is computed in Proposition 2 in Appendix B.

If more such low Hamming weight vectors are available, then the complexity of the attack may be further reduced, although it seems unlikely to obtain a significant advantage over exhaustive search with this approach.

A.1.2 Parameter Selection for the $(2,3)$-OWF.

According to the analysis, we determine parameters n, m, t for which we conjecture that the $(2,3)$-OWF has s bits of security.

First, due to the exhaustive search, we require $n \geq s$. Second, the most restrictive constraint on m is imposed by the subset-sum algorithm (Section 4). If we conservatively ignore the hidden polynomial factors and the large memory complexity of the subset-sum algorithm of [17], we need to set

$$0.283m \geq s.$$

Overall, we obtain

$$n = \log 3 \cdot t = s,$$

and

$$m = \tfrac{s}{0.283} \approx 3.53s.$$

We now consider the attack exploiting low Hamming weight combinations of the rows of $\mathbf{B}$, and in particular, Proposition 2. In our case, we apply the proposition with $\log 3 \cdot t = n$ and $\ell = n$. For $m = 3.53n$, we obtain that the probability of having a vector of Hamming weight at most n is bounded by

$$2 \cdot 2^{m(H(0.283)+2\cdot 0.283-\log 3)} \approx 2 \cdot 2^{-0.16m} \approx 2 \cdot 2^{-0.56s}.$$

For $s \geq 128$, the expression evaluates to (less than) 2^{-70}, so it is unlikely to encounter such an event in practice. Moreover, even if the event occurs, security only regrades by a factor of 3, and by the same proposition the probability that two such vectors are spanned by the rows of B is at most 2^{-140}. Nevertheless, one may increase n (and correspondingly t) by a few bits (at negligible cost) to defeat this attack vector completely.

A more aggressive setting of the parameters may take into account the polynomial factors of [17] (and perhaps its high memory complexity). Unfortunately, the polynomial factors associated with the complexity formulas of the relevant subset-sum algorithms have not been analyzed. For example, if we assume that the polynomial factors are about m^2, and we aim for $s = 128$ bits of security, then we require $m^2 \cdot 2^{0.283m} \geq 2^s = 2^{128}$. Setting $m = 400 = 3.125s$ is sufficient for satisfying the constraint in this setting.

A.2 Security Evaluation of the $(2,3)$-wPRF

As for the $(2,3)$-OWF, we describe several attacks and analyze their complexity as a function of the parameters n, m, t.

Unlike the case of the $(2,3)$-OWF (where the goal was to find a preimage of a give output), we can choose a small value of t regardless of the other parameters without sacrificing security. In fact, it is clear that a small value of t can only contribute to security, as any attack on a scheme with a small value of t can be applied to a scheme with a larger value of t, simply by ignoring part of the output. Consequently, we may fix t to the smallest value acceptable by the application.

We also note that given a sufficiently large number of samples, we expect that the key k would be uniquely determined by the samples (regardless of the value of t).

A.2.1 Key recovery attacks exploiting a few samples.

We describe key recovery attacks that make use of the minimal number of samples required to derive the secret key k.

First, as for the $(2,3)$-OWF, exhaustive search requires 2^n time. Also, similarly to the $(2,3)$-OWF, given any sample $(x, \mathbf{B}, y)$, we can guess $m - t$ bits of $w = \mathbf{K}x$ and solve the linear equation system $y = \mathbf{B}w \bmod 3$, which then allows to compute a suggestion for k (that can be tested on the remaining samples). This attack has complexity 2^{m-t}.

Furthermore, given a single sample, we can apply the same attack for enumerating w values for the $(2,3)$-OWF. This attack has complexity which is the maximum between $2^{m/2}$ and $2^{m-\log 3\cdot t}$.

Reduction to subset-sum. As for the $(2,3)$-OWF, we can reduce the key recovery problem to the problem of solving subset-sum over the m binary variables of w. However, it is clear that if the algorithm is applied to a single $(x, \mathbf{B}, y)$ sample, then its expected complexity cannot drop below $2^{m-\log 3 \cdot t-(m-n)} = 2^{n-\log 3 \cdot t}$, which is the expected number of w values possible given $(x, \mathbf{B}, y)$ (the remaining key candidate after analyzing one sample are tested against another sample).

On the other hand, if we try to reduce the complexity by applying the algorithm to more than one sample (e.g., to $(x, \mathbf{B}, y)$ and $(x', \mathbf{B}, y')$), then we must take advantage of the dependency between $w = \mathbf{K}x$ and $w' = \mathbf{K}x'$, which are related via linear constraints, imposed by k and by x, x'. However, it is not clear how to model these complex linear constraints in the subset-sum reduction and we were not able to improve the complexity of the single-sample attack.

A.2.2 Exploiting multiple samples.

The key recovery attacks described above make use of a minimal number of samples required to derive the secret key. On the other hand, when given more samples, it may be possible to exploit various relations among them to mount distinguishing and even key recovery attacks which we investigate below.

Output bias. We consider a single sample $(x, \mathbf{B}, y)$ and analyze the bias of linear combinations of the entries of y over $\mathbb{Z}_3$. If any such linear combination has a sufficiently high bias towards some constant, then an attacker can exploit it in a distinguishing attack.

Similarly to the case analyzed for the corresponding $(2,3)$-OWF, assume there are vectors $v \in \mathbb{Z}_3^m$ and $u \in \mathbb{Z}_3^t$ such that $u\mathbf{B} = v$ and the Hamming weight of v is ℓ. Given $y = \mathbf{B}w \bmod 3$, the attacker computes $uy \bmod 3 = vw \bmod 3$ and thus obtains the value of a linear combination $\bmod 3$ of ℓ entries of $w \in \{0,1\}^m$. Specifically, denoting the set of ℓ non-zero indices of v by I, the attacker computes $\sum_{i \in I} v_i w_i \bmod 3$. We now calculate the bias of sum the mod 3, assuming that w is uniformly distributed in $\mathbb{Z}_2^m$.

Each non-zero coefficient of the linear combination v is either 1 and 2. It is easy to prove by induction on ℓ (or by analysis of sums of binomial coefficients) that for any coefficients $v_i \in \{1,2\}$ where $i \in I$ and for any $a \in \{0,1,2\}$,

$$\Pr\left[\sum_{i \in I} v_i w_i \bmod 3 = a\right] \in \{\tfrac{1}{3} \pm \tfrac{1}{2^\ell}, \tfrac{1}{3} \pm \tfrac{2}{2^\ell}\},$$

where the actual probability depends on v, a and ℓ. Thus, the bias of $vw \bmod 3$ is bounded by $\frac{2}{2^\ell}$.

We use Proposition 2 to deduce that the subspace spanned by the rows of $\mathbf{B}$ contains a vector of Hamming weight at most ℓ with probability at most $2 \cdot 2^{m(H(\ell/m)-\log 3)+\ell+\log 3 \cdot t}$.

Our analysis is conservative, as we ignore the work performed by the attacker to find a low Hamming weight vector v spanned by the rows of B. Thus, we will be interested in $\ell \approx s/2$, as we would like to avoid having a linear combination of the output with bias at least $2^{-s/2}$ (except with small probability). Plugging this into the formula above we bound the probability by

$$2 \cdot 2^{m(H(s/2m)-\log 3)+s/2+\log 3 \cdot t}. \tag{2}$$

Conditional output bias. As described above, the bias of expressions of the form $\sum_{i\in I} v_i w_i \bmod 3$, (where $I \subseteq [m]$ is a set of size ℓ, each $v_i \in \{1,2\}$ is fixed and $w_i \in \{0,1\}$ are independent and uniformly distributed random variables) is bounded by $2^{-\ell+1}$.

However, the bias may increase if information about the variables w_i is known. In particular, the case in which $\sum_{i\in I} w_i \bmod 2$ is known was analyzed in [31], where the authors showed that the conditional biases such as

$$\left|\Pr\left[\sum_{i\in I} w_i \bmod 3 = 0 \mid \sum_{i\in I} w_i \bmod 2 = 0\right] - 1/3\right|$$

can be as large as about $2^{-0.21\ell}$. While this is still exponentially small in ℓ, it is more significant than the unconditional bias.

The PRF candidate proposed in [15] and analyzed in [31] is similar to the one we analyze for $n = m$, yet the matrix $\mathbf{B}$ contains a single row that sums mod 3 all the entries of w. Moreover, since $\mathbf{K}$ is a circulant matrix and x has even Hamming weight, then $\sum_{i\in[m]} w_i \bmod 2 = 0$ and the distinguishing attack is applicable. In our case, $\mathbf{B}$ is selected at random and the parameters are chosen such that the attacker cannot obtain $\sum_{i\in I} w_i \bmod 3$ for any fixed set I (and particularly $I = [m]$), except with negligible probability. The general distinguishing attack is analyzed below.

Assume that given a sample $(x, \mathbf{B}, y)$, there exists a set $I \subseteq [m]$ (that depends of x) such that $\sum_{i\in I} w_i \bmod 2$ is known to the attacker. Moreover, assume that there exists v in the row span of $\mathbf{B}$ such that $v_i = 1$ for each $i \in I$ and $v_i = 0$ for each $i \notin I$. Then, the value $\sum_{i\in I} w_i \bmod 3$ can be computed from the output, and the distinguishing advantage is as high as (about) $2^{-0.21\ell}$. Of course, if several samples are available, the distinguishing advantage can increase by accumulating the bias, assuming the above conditions are fulfilled for more than one sample.

An extended variant of the attack may consider a vector $v' = v + u$ in the row span of $\mathbf{B}$. Denoting that Hamming weights of v and u by ℓ_1,ℓ_2, respectively, the conditional bias can be as high as $2^{-0.21\ell_1-\ell_2+1}$. Note that here we conservatively ignore the work required to find such v'.

Given 2^r samples (where $r \leq 40$), we wish to show that the distinguishing advantage is small (except with negligible probability). Indeed, calculation shows that this distinguisher is not stronger that the unconditional distinguisher. Essentially, given a single sample, for any fixed I, the vector v as described above is in the row span of $\mathbf{B}$ with minuscule probability 3^{t-m}. In the extended attack, we consider a ball around v, but this ball is much smaller than the one considered in the unconditional distinguisher (as $\ell_2 < s/2$).

Finally, in a more general variant of the attack, the attacker may guess a parity of key bits and calculate the conditional bias over several samples, attempting to amplify it. Note that the desired vector v changes for each sample according to x. By similar calculation, once the attacker has fixed the guess given a sample, almost all additional samples would not allow to amplify the conditional bias, as a good vector is unlikely to be in the row span of $\mathbf{B}$.

Differential cryptanalysis and low Hamming weight samples. As we argue below, the $(2, 3)$-wPRF seems to be immune to classical statistical differential cryptanalysis.

Assume that the attacker obtains two samples $(x, \mathbf{B}, y)$ and $(x + \delta \bmod 2, \mathbf{B}, y')$, where $\delta \in \{0, 1\}^n$. Denote $w = \mathbf{K}x \bmod 2$ and $w' = \mathbf{K}(x + \delta) \bmod 2 = w + \mathbf{K}\delta \bmod 2$. Thus, $y' = \mathbf{B}w' \bmod 3 = \mathbf{B}(w + \mathbf{K}\delta \bmod 2) \bmod 3$. In general, y and y' do not seem to have any statistical relation that holds with sufficiently high probability. Particularly, $w + \mathbf{K}\delta \bmod 2 \neq w + \mathbf{K}\delta \bmod 3$, except with very small probability.

From an algebraic viewpoint, the attacker can consider the m bits of $w = Kx \bmod 2$ as variables over $\mathbb{Z}_2$, but then the algebraic degree of the output over $\mathbb{Z}_3$ would be large. The attacker can also consider the m bits of w as variables over $\mathbb{Z}_3$. In this case, the attacker obtains t linear equations and t quadratic equations mod 3 (of the form $(w_i)^2 - w_i = 0$) from the sample $(x, \mathbf{B}, y)$. On the other hand, the algebraic degree of $w' = \mathbf{K}(x + \delta) \bmod 2$ in the variables of w would be large due the dense mod 2 operations, hence it is not clear how to obtain additional low degree equations. In general, such algebraic attacks do not seem more efficient than the attack described above for enumerating w values.

Another scenario which we consider is when the attacker obtains a sample $(x, \mathbf{B}, y)$ such that x is of low Hamming weight. In this case each entry of w is a low Hamming weight sum $\bmod 2$ of the bits of k and can thus be described as a low degree polynomial over $\mathbb{Z}_3$. Consequently, low degree polynomial equations over $\mathbb{Z}_3$ in the secret key can be deduced from the output. Obtaining several such samples may allow the attacker to solve for the key. In general, such low Hamming weight samples are avoided with high probability given the data limit, but we can also place a restriction on the sample generation, forcing it to generate vectors x with minimal Hamming weight (e.g., a lower bound of $n/4$).

Attacks that exploit the structure of K. Recall the $\mathbf{K}$ is a circulant matrix and we consider attacks that exploit its special structure. A notable property of $\mathbf{K}$ as a circulant matrices is that it preserves symmetry. Specifically, if $x \in \mathbb{Z}_2^n$ is 2-symmetric (i.e., its two halves are equal), then $w = \mathbf{K}x$ is also 2-symmetric. Thus, given a sample $(x, \mathbf{B}, y)$ such that x is 2-symmetric, we know that w is 2-symmetric and this fact can be used to uniquely recover w (which is effectively of length $m/2$) in time $2^{n-m/2-\log 3 \cdot t}$. Consequently, we can derive $m/2$ linear equations mod 2 on the secret key and perhaps use another sample to recover it completely.

The probability that such a 2-symmetric vector is found in the data is $2^{-n/2+r}$. We require that this quantity is negligible.

It is also possible to consider even more symmetry, such as 4-symmetric vectors, but these have lower probability of being picked.

Finally, we leave it to future work to extend this attack beyond the simple case analyzed above. For example, one may consider input vectors that are close to 2-symmetric, or pairs of samples $(x, \mathbf{B}, y), (x', \mathbf{B}, y')$ such that $x + x' \bmod 2$ is 2-symmetric (implying that $\mathbf{K}(x + x')$ is 2-symmetric).

Attacks based on self-similarity. There are several simple attacks that take advantage of the fact that $\mathbf{B}$ is fixed for all samples generated with the same secret k. A basic attack looks for a collision on the w values for a pair of samples, which can be detected at the output. Given 2^r samples, the advantage of this attack is $2^{2r - rank(\mathbf{K})}$. we will set the parameters such that the advantage of this attack is negligible for the data complexity bound (assuming the rank of K is not too small).

Another simple attack is a multi-target attack, where given 2^r samples, the attacker guesses w, computes $\mathbf{B}w \bmod 3$ and compares the result with all given outputs. A match allows to recover a candidate for the secret k. The expected complexity of this attack is 2^{m-r}, but cannot drop below $2^{m - \log 3 \cdot t}$ without exploiting relations between the different w values. In general, given the data limit, the attack less efficient that the attack that enumerates all w values considered above.

Simultaneous sums. We describe a more involved self-similarity attack, which exploits the fixed $\mathbf{B}$ value per secret k.

Assume that for some index set I of size at least 3, $\sum_{i \in I} x^{(i)} \bmod 2 = 0$. Then, $\sum_{i \in I} w^{(i)} \bmod 2 = 0$. While it is not clear how this relation influences the output, we extend this initial observation by simultaneously considering sums mod 2 and 3, as follows: assume that there are 4 samples (denoted for simplicity by $\{(x^{(i)}, \mathbf{B}, y^{(i)})\}_{i=1}^{4}$) such that for each $j \in [m]$,

$$\sum_{i=1}^{4} w_j^{(i)} = 2.$$

Then, $\sum_{i=0}^{4} x^{(i)} \bmod 2 = 0$ and $\sum_{i=0}^{4} y^{(i)} \bmod 3 = B \cdot \mathbf{2} \bmod 3$ (where $\mathbf{2}$ is a vector with m entries whose values are 2). Note that a random 4-tuple of samples satisfies this simultaneous sum constraint with probability $2^{-n - \log 3 \cdot t}$, but the probability that the constraint $\sum_{i=0}^{4} w_j^{(i)} = 2$ is satisfied for every $j \in [m]$ is about

$$\left(\frac{\binom{4}{2}}{16} \right)^{-m} \approx 2^{-1.415m},$$

which is higher than expected if $1.415m < n + \log 3 \cdot t$.

Since the adversary has about 2^{4r} such 4-tuples, the probability that such a simultaneous 4-sum exists is about $2^{4r - 1.415m}$. It can be detected in time complexity of about 2^{2r} using a standard matching algorithm. The important constraints for defending against this attack are

$$1.415m > n + \log 3 \cdot t,$$

or

$$2^{4r-1.415m} \tag{3}$$

is negligible, otherwise.

The simultaneous 4-sum distinguisher can be easily generalized to a simultaneous d-sum distinguisher for arbitrary d. In general, we look for d-tuples where (for example) for each $j \in [m]$,

$$\sum_{i=1}^{d} w_j^{(i)} = c,$$

for some value c (fixed mod 6) such that $c \bmod 2 = 0$. However, calculation shows that $d = 4$ gives the most efficient distinguisher in our case (with the small data limit).

A.2.3 Parameter selection for the $(2,3)$-wPRF.

According to the analysis, we determine parameters n, m, t for which we conjecture that the $(2,3)$-wPRF has s bits of security.

In order to select the parameters, we may first set t to it's minimal possible value (depending on the application). We also assume that t is not too large, and particularly $\log 3 \cdot t \leq s$.

The constraints imposed by the above attacks are as follows. First, due to exhaustive search, we require

$$n \geq s.$$

Second, the subset-sum algorithm imposes the constraint $n - \log 3 \cdot t \geq s$, (given that $n \geq m$ and $\log 3 \cdot t \leq s$).

We consider two sets of parameter. The first is

$$n = m = s + \log 3 \cdot t.$$

In particular, if $\log 3 \cdot t = s$, then $n = m = 2s$. The second parameter set

$$n = m = 1.25(s + \log 3 \cdot t),$$

and is less aggressive.

Next, we analyze the bias of the output based on (2), assuming that $\log 3 \cdot t = s$. For the first parameter set, we obtain that the probability of having bias of $\frac{2}{2^{s/2}}$ is bounded by

$$2 \cdot 2^{m(H(s/2m) - \log 3) + s/2 + \log 3 \cdot t} = 2 \cdot 2^{2s(H(1/4) - \log 3) + 3s/2} \approx 2 \cdot 2^{-0.049s},$$

which is non-negligible. On the other hand, the consequences of the distinguishing attack given the data limit seem relatively mild, and this parameter set may be considered by applications where performance is critical.

For the second parameter set (assuming $\log 3 \cdot t = s$), the probability of having bias $\frac{2}{2^{s/2}}$ is bounded by

$$2 \cdot 2^{m(H(s/2m)-\log 3)+s/2+\log 3 \cdot t} = 2 \cdot 2^{2.5s(H(1/5)-\log 3)+3s/2} \approx 2 \cdot 2^{-0.65s},$$

which we consider negligible.

The advantage of the collision attack is $2^{2r-rank(\mathbf{K})} \leq 2^{80-rank(\mathbf{K})}$. Given that $m = 2s \geq 256$, if we choose n as a power of 2, based on Proposition 1, the advantage is much smaller than $2^{s/2}$ (except with negligible probability).

Finally, we consider the attack that exploits the circulant matrix $\mathbf{K}$ and the simultaneous 4-sum distinguisher. For defending against the first attack, when $s \geq 128, r \leq 40$, we have $2^{-n/2+r} < 2^{-128+40} = 2^{-88}$ ($n \geq 2s$), which we consider negligible.

Recall that the probability of having such a 4-sum in the output is estimated in (3) as $2^{4r-1.415m} \leq 2^{160-2.83s}$. For a minimal choice of $s = 128$, this probability is smaller than 2^{-200} which is negligible.

A.3 Security Analysis of the LPN-PRG

We analyze the security of the LPN-PRG.

If the matrix $\mathbf{A}$ was a random matrix, then the first step would consist of generating m samples from the alternative weak PRF construction proposed in [15]. Each sample $w_i = \mathbf{A}[i]x \bmod 2 + ((\mathbf{A}[i]x \bmod 3) \bmod 2) \bmod 2 \in \mathbb{Z}_2^m$ can be viewed as adding noise $((\mathbf{A}[i]x \bmod 3) \bmod 2)$ to the inner product $\mathbf{A}[i]x$ mod 2. Given that $\mathbf{A}[i]$ is of sufficiently large Hamming weight, then $\Pr_x[(\mathbf{A}[i]x \bmod 3) \bmod 2 = 1] \approx 1/3$, which is the magnitude of noise added. The second step consists of a compressing linear transformation B applied to w. The idea is to increase the noise of each sample by mixing it with other samples. This step should defeat standard attacks applied to LPN with a constant noise parameter (such as decoding attacks).

In our case, the matrix $\mathbf{A}$ is structured (it is a random Toeplitz matrix), but we were not able to exploit this in an efficient attack.

A.3.1 Key recovery attacks.

We begin by considering attacks that attempt to recover the secret key (seed).

Exhaustive search for the key recovery attacks requires time 2^n.

In a different approach for recovering the key, given a sample $\mathbf{A}, \mathbf{B}, y$, the attacker enumerates over the subspace of w values that satisfy $\mathbf{B}w = y \bmod 2$. This subspace contains 2^{m-t} vectors. For each such vector, the attacker attempts to recover x given $\mathbf{A}$ and w. Thus, given $\mathbf{A}, w$ the attacker has m samples generated from the LPN-like construction proposed in [15] as a weak PRF (although we a structured matrix $\mathbf{A}$). Given that m is not too large (i.e., it is a small multiple of n), then the best attack we have on this scheme simply tries to break the LPN instance (which has noise of 1/3), without exploiting the deterministic way in which it is generated. The concrete security of LPN given a small number was analyzed in several publications such as [39], and the complexity of known attack is generally exponential in n. Nevertheless, the attacker is required to solve 2^{m-t} related LPN instances, and perhaps can amortize the complexity. Moreover, the matrix $\mathbf{A}$ is structured. Thus, we (conservatively) estimate the total complexity of such attacks by 2^{m-t}.

A.3.2 Noisy linear equations.

As noted above, each bit w_i for $i \in [m]$ can be viewed as a noisy linear equation over $\mathbb{Z}_2$ with noise of about $1/3$, or bias $2/3 - 1/2 = 1/6$. Our goal is to select the parameter m such that the all linear combinations of the output bits of y have exponentially small bias towards a linear equation in the secret x. We will (heuristically) model each bit w_i as having independent bias of $1/6$.

If the linear subspace spanned by the rows of $\mathbf{B}$ contains a vector of Hamming weight ℓ, then by the piling-up lemma, the bias of the corresponding linear combination of the bits of w is

$$2^{\ell-1} \cdot (\tfrac{1}{6})^{\ell} < 2^{-\log 3 \cdot \ell}. \tag{4}$$

Thus, we will require that the rows of $\mathbf{B}$ do not span a vector whose Hamming weight is too low. This is similar to the previously analyzed schemes, but the lower bound on the Hamming weight we will enforce for the PRG will be lower. Indeed, the bias above is calculated with respect some linear equation in the unknown secret key bits. Such a bias is generally much less of a security concern compared to a bias towards a constant value (e.g., the bias analyzed for the $(2,3)$-wPRF construction) which can be used to directly distinguish the output from random using statistical tests. Particularly, an alternative scheme where we change the first transformation to only compute the "noise part" ($w = (\mathbf{A}x \bmod 3) \bmod 2$) would require larger parameters to be secure, as we need a higher lower bound on the Hamming weight ℓ to avoid distinguishing attacks.

For a PRG with s bits of security, we will conservatively require a bias of at most $2^{-0.1s}$. We are not aware of any attack that can exploit such a low bias. Effectively, this means that the minimal distance of B should be at least $s \cdot 0.1/\log 3 < 0.07s$ (except with small probability).

Remark 1. In [31] the authors analyze the constructions presented in [15]. In particular, for the alternative PRF construction, given a sample $(a \in \mathbb{Z}_2^n, x \cdot a)$, they show that there exists $j \in [n]$ such that

$$|\Pr[a_j = 0 \mid x_j = 0 \text{ and } (a \cdot x \bmod 2 + ((a \cdot x \bmod 3) \bmod 2) \bmod 2) = 0]| \approx \tfrac{1}{2^{0.21\ell}},$$

where ℓ is the Hamming weight of a. This property was exploited in a distinguishing attack.

Our PRG construction seems to be immune to this type of analysis because the attacker only has access to sufficiently dense linear combinations of (structured) samples of the alternative PRF construction.

A.3.3 Parameter selection for the LPN-PRG.

We determine parameters n, m, t for which we conjecture that the PRG has s bits of security.

Recall the we set $t = 2n$. Exhaustive search implies that $n \geq s$ and we have lower bounded the effort required in key recover by 2^{m-t}. Thus, a reasonable choice of parameters is $n = s$ and $m = 3s, t = 2s$.

For these parameters, we consider the maximal bias of linear combinations according to (4), with $\ell = 0.07s$. By Proposition 2, the probability of having a vector of Hamming weight more than $0.07s$ in the row span of B is $0.07s \cdot 2^{3s(H(0.07/3)-1)+2s} < 0.07s \cdot 2^{-0.52s}$. We consider this as a negligible probability as the consequences of the this unlikely event are mild.

A.4 Security analysis of LPN-PRF

The overall structure of the LPN-PRF is similar to the PRG. However, unlike PRG, in the first transformation, the key is part of the matrix and x is a public value. Thus, w can be viewed as m outputs of (another) more structured version of the construction of [15].

In terms of parameters, the main differences are that we have $n \geq m$ and t is not constraint to $2n$ (in fact, we will propose to set it smaller than n). Furthermore, we assume that the attacker obtains 2^r samples for $r \leq 40$ instead of a single sample.

We note that variants of the basic attacks that were analyzed for the $(2,3)$-wPRF (such as the attack that exploit the circulant (K), the collision attack and the multi-target attack) are also applicable to this construction. As in the case of the $(2,3)$-wPRF, they do not seem to be a threat for our choice of parameters.

Overall, we have not found any class of attacks that are applicable to this construction, but not to the previous ones. Below we briefly consider the most important attacks that influence the choice of parameters.

Summary of attacks. As for the PRG, we estimate the total complexity of key recovery attacks by 2^{m-t}. Although we have more samples that for the PRF, it is not clear how to exploit them to obtain better complexity.

In addition, similarly to the PRG, we require that each linear combination of the output has bias of at most $2^{-0.1s}$ (toward some linear combination of the key over $\mathbb{Z}_2$).

Parameter selection for the LPN-PRF. For s-bit security, we set $n = m = 2s$ and $t = s$. By our analysis, exhaustive search requires $2^{m-t} = 2^s$ time.

We consider the maximal bias of linear combinations according to (4), with $\ell = 0.07s$. By Proposition 2, the probability of having a vector of Hamming weight more than $0.07s$ in the row span of $\mathbf{B}$ is $0.07s \cdot 2^{2s(H(0.07/2)-1)+s} < 0.07s \cdot 2^{-0.56s}$, which we consider negligible.

B Analysis of Random Linear Codes

We analyze the distance of random linear codes defined by a random matrix or a random Toeplitz matrix. The analysis is based on the probabilistic method and is similar to the analysis used to obtain the Gilbert–Varshamov bound.

Proposition 2. *Let* $\mathbf{B} \in \mathbb{Z}_q^{t \times m}$ *be a random matrix, or a random Toeplitz matrix whose rows define a linear code. Then, the minimal distance of (the code defined by)* $\mathbf{B}$ *is at most* $\ell < m/2$ *with probability at most*

$$f_q(m, t, \ell) \leq q^{t-m} \cdot Vol_q(m, \ell),$$

where $Vol_q(m, \ell) = \sum_{i=1}^{\ell} \binom{m}{i} \cdot (q-1)^i$*. Moreover, this code contains two such linearly independent vectors with probability at most* $(f_q(m, t, \ell))^2$*.*

Finally, let $H(p) = -p \log p - (1-p) \log(1-p)$ *be the binary entropy function. Then*

$$f_2(m, t, \ell) \leq \ell \cdot 2^{m(H(\ell/m)-1)+t},$$

and

$$f_3(m, t, \ell) \leq 2 \cdot 2^{m(H(\ell/m)-\log 3)+\ell+\log 3 \cdot t}.$$

Proof. The number of non-zero vectors of Hamming weight at most ℓ over $\mathbb{Z}_q^m$ is $Vol_q(m, \ell) = \sum_{i=1}^{\ell} \binom{m}{i} \cdot (q-1)^i$.

For $q = 2$, we have $Vol_2(m, \ell) = \sum_{i=1}^{\ell} \binom{m}{i} < \ell \cdot \binom{m}{\ell} \leq \ell \cdot 2^{mH(\ell/m)}$, while for $q = 3$, we have $Vol_3(m, \ell) = \sum_{i=1}^{\ell} \binom{m}{i} 2^i < 2 \cdot \binom{m}{\ell} 2^\ell \leq 2 \cdot 2^{mH(\ell/m)+\ell}$.

Since $\mathbf{B}$ is selected uniformly from a pairwise independent hash family (either a random matrix or a random Toeplitz matrix), the probability that every non-zero vector is in the row space is $q^{t-m} - 1 < q^{t-m}$. By a union bound over all vectors of Hamming weight bounded by ℓ, the probability that there exists such a vector in the row space of B is at most

$$f_q(m, t, \ell) \leq q^{t-m} \cdot Vol_q(m, \ell).$$

The bound $(f_q(m, t, \ell))^2$ on the probability of having two such linearly independent vectors follows by pairwise independence.

Specifically, for $q = 2$, we obtain $f_2(m, t, \ell) \leq \ell \cdot 2^{m(H(\ell/m)-1)+t}$, while for $t = 3$, we obtain $f_3(m, t, \ell) \leq 2 \cdot 3^{t-m} \cdot 2^{mH(\ell/m)+\ell} = 2 \cdot 2^{m(H(\ell/m)-\log 3)+\ell+\log 3 \cdot t}$.

C Deferred Protocol Details

Circuit description of constructions. We can represent our constructions by circuits consisting of the gates described previously. This approach is similar to that of [20]. We formally define computation circuit representations for our constructions in Definition 4.

Definition 4 (Computation circuit). *A computation circuit C with input space $\mathbb{G}^{\mathit{in}} = \prod \mathbb{G}_i^{\mathit{in}}$ and output space $\mathbb{G}^{\mathit{out}} = \prod \mathbb{G}_i^{\mathit{out}}$ is a (labeled) directed acyclic graph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes the set of vertices and $\mathcal{E}$ denotes the set of edges according to the following:*

- *Each source vertex corresponds to exactly one $\mathbb{G}_i^{\mathit{in}}$ and vice versa. The label for the vertex is the identity function on the corresponding $\mathbb{G}_i^{\mathit{in}}$. Each sink vertex corresponds to exactly one $\mathbb{G}_i^{\mathit{out}}$ and vice versa. The label for the vertex is the identity function on the corresponding $\mathbb{G}_i^{\mathit{out}}$. Each non-source $V \in \mathcal{V}$ is labeled with a gate $\mathcal{G}_V \in \mathsf{Gates}$ that computes the function $\mathcal{G}_V : \mathbb{G}_V^{\mathit{in}} \to \mathbb{G}_V^{\mathit{out}}$. The depth of a vertex $V \in \mathcal{V}$, denoted by $\mathit{depth}(V)$ is the length of the largest directed path from a source vertex to V.*
- *For an edge (V_a, V_b), let $\mathbb{G}_{V_a}^{\mathit{out}} = \prod \mathbb{G}_{V_a,i}^{\mathit{out}}$ and $\mathbb{G}_{V_b}^{\mathit{in}} = \prod \mathbb{G}_{V_b,i}^{\mathit{in}}$. Then, there exists indices j and k such that $\mathbb{G}_{V_a,j}^{\mathit{out}} = \mathbb{G}_{V_b,k}^{\mathit{in}}$. Further, for each input $\mathbb{G}_{V_b,i}^{\mathit{in}}$ for V_b, there is some edge (V_c, V_b) that satisfies the above.*
- *The evaluation of the gate for vertex V on input $x \in \mathbb{G}_V^{\mathit{in}}$ is defined as $y = \mathcal{G}_V(x)$. The evaluation of the circuit C, denoted by $\mathit{Eval}_C(x)$, where $x \in \mathbb{G}^{\mathit{in}}$ is the value $y \in \mathbb{G}^{\mathit{out}}$, that is obtained by recursively evaluating each gate function in the circuit.*

Let $\mathsf{F} = \{\mathsf{F}_\lambda\}_{\lambda \in \mathbb{N}}$ denote a family of functions $\mathsf{F}_\lambda : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. We say that $\{C_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of computation circuits for F if all C_λ have the same topological structure, and for all $\lambda \in \mathbb{N}$, $\mathsf{F}_\lambda(x) = \mathit{Eval}_C(x)$ for all $x \in \mathcal{X}_\lambda$.

C.1 Local Protocols for Circuit Gates

Here, we provide the remaining details for the circuit gate protocols.

Protocol notation and considerations. For a protocol $\boldsymbol{\pi}$, we use the notation $\boldsymbol{\pi}(a_1, \ldots, a_k \mid b_1, \ldots, b_l)$ to denote that the values $a_1, \ldots, a_k$ are provided publicly to all parties in the protocol, while the values $b_1, \ldots, b_l$ are secret shared among the parties. When P_i knows the values $(a_1, \ldots, a_k)$, and has shares $[\![b_1]\!]^{(i)}, \ldots, [\![b_l]\!])$, we use the notation $\boldsymbol{\pi}(a_1, \ldots, a_k \mid [\![b_1]\!]^{(i)}, \ldots, [\![b_l]\!]^{(i)})$ to denote that P_i runs the protocol with its local inputs.

Given public values $a_1, \ldots, a_k$, it is straightforward for the protocol parties to compute a sharing $[\![f(a_1, \ldots, a_k)]\!]$ for a function f (for example, P_1 computes the function as its share, and all other parties set their share to 0).

Linear gate protocol $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{A},p}$. The linear gate is the easiest to evaluate, and follows from the standard linear homomorphism of additive secret sharing.

- **Functionality**: Each party is provided with the matrix $\mathbf{A}$ and shares of the input x (over $\mathbb{Z}_p$). The goal is to compute shares of the output $y = \mathbf{A}x$.
- **Preprocessing**: None required.
- **Protocol details**: For the protocol $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{A},p}(\mathbf{A} \mid x)$, each party P_i computes its output share as $[\![y]\!]^{(i)} = \mathbf{A}\,[\![x]\!]^{(i)}$. Note that this works because $\mathbf{A}x = \sum_{\mathrm{P}_i \in \mathcal{P}} \mathbf{A}\,[\![x]\!]^{(i)}$ as a direct consequence of the linear homomorphism of additive shares.

Addition gate protocol $\boldsymbol{\pi}^{\boldsymbol{p}}_{\mathsf{Add}}$. The addition modulo p gate is also easy to evaluate. Given shares of x, x' over $\mathbb{Z}_p$, for the protocol each party P_i can locally compute its share of $x + x'$ as $[\![x]\!]^{(i)} + [\![x']\!]^{(i)} \bmod p$. This directly follows from the additive homomorphism of additive shares.

Bilinear gate protocol $\boldsymbol{\pi}^{\boldsymbol{p}}_{\mathsf{BL}}$. The bilinear gate protocol is essentially a generalization of Beaver's multiplication triples [8, 20] that computes the multiplication of two shared inputs. For Beaver's protocol, to compute a sharing of ab given shares of a and b (all sharings are over a ring $\mathcal{R}$), the protocol parties are provided shares of a randomly sampled triple of the form $(\tilde{a}, \tilde{b}, \tilde{a}\tilde{b})$ in the preprocessing stage. Beaver's protocol first reconstructs the masked inputs $\hat{a}$ and $\hat{b}$ after which local computation is enough to produce shares of the output. For our bilinear gate protocol, we assume that all parties are already provided with the masked inputs (to move the communication outside of the gate protocol), along with correlated randomness similar to a Beaver triple.

- **Functionality**: Abstractly, the goal of the bilinear gate protocol is to compute shares of the output $y = \mathbf{K}x$ given shares of both inputs $\mathbf{K}$ and x. For our purpose however, the masked inputs will have already been constructed beforehand, i.e., each party is provided with $\hat{\mathbf{K}}$ and $\hat{x}$ publicly, along with shares of correlated randomness similar to a Beaver triple (see below).
- **Preprocessing**: Each party is provided shares of $\tilde{\mathbf{K}}, \tilde{x}$, and $\tilde{\mathbf{K}}\tilde{x}$ as correlated randomness.
- **Protocol details**: For the protocol $\boldsymbol{\pi}^{p}_{\mathsf{BL}}(\hat{\mathbf{K}}, \hat{x} \mid \tilde{\mathbf{K}}, \tilde{x}, \tilde{\mathbf{K}}\tilde{x})$, each party P_i computes its share of $\hat{y}$ as:

$$[\![\hat{y}]\!]^{(i)} = \left[\!\!\left[\hat{\mathbf{K}}\hat{x}\right]\!\!\right]^{(i)} - \hat{\mathbf{K}}\,[\![\tilde{x}]\!]^{(i)} - \left[\!\!\left[\tilde{\mathbf{K}}\right]\!\!\right]^{(i)}\hat{x} + \left[\!\!\left[\tilde{\mathbf{K}}\tilde{x}\right]\!\!\right]^{(i)}$$

  Correctness. Note that this works since:

$$\begin{aligned}\sum_{\mathrm{P}_i \in \mathcal{P}} [\![\hat{y}]\!]^{(i)} &= \hat{\mathbf{K}}\hat{x} - \hat{\mathbf{K}}\tilde{x} - \tilde{\mathbf{K}}\hat{x} + \tilde{\mathbf{K}}\tilde{x} \\ &= (\mathbf{K} + \tilde{\mathbf{K}})x - \tilde{\mathbf{K}}(x + \tilde{x}) + \tilde{\mathbf{K}}\tilde{x} \\ &= \mathbf{K}x\end{aligned}$$

Since the output of the bilinear gate will usually feed into a conversion gate which requires the input to be already masked, as an optimization, we can have the bilinear gate itself compute shares of the masked output, i.e., $\hat{y} = \mathbf{K}x + \tilde{y}$. This can be done by providing the correlated randomness $\tilde{\mathbf{K}}\tilde{x} + \tilde{y}$ instead of $\tilde{\mathbf{K}}\tilde{x}$. The upshot of this optimization is that one fewer piece of correlated randomness will be required.

$\mathbb{Z}_2 \to \mathbb{Z}_3$ conversion protocol $\boldsymbol{\pi}^{(\mathbf{2},\mathbf{3})}_{\mathsf{Convert}}$.

- **Functionality**: Abstractly, the goal of the $\mathbb{Z}_2 \to \mathbb{Z}_3$ conversion protocol is to convert a sharing of x over $\mathbb{Z}_2$ to a sharing of the same $x^* = x$, but now over $\mathbb{Z}_3$. For our purpose, the parties will be provided the masked input $\hat{x} = x \oplus \tilde{x}$ (i.e., masking is over $\mathbb{Z}_2$) directly along with correlated randomness that shares $\tilde{x}$ over $\mathbb{Z}_3$.

- **Preprocessing**: Each party is also provided with shares of the mask $r = \tilde{x}$ over $\mathbb{Z}_3$ as correlated randomness.
- **Protocol details**: For the protocol $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}(\hat{x} \mid r)$, each party proceeds as follows:
$$[\![x^*]\!]^{(i)} = [\![\hat{x}]\!]^{(i)} + [\![r]\!]^{(i)} + (\hat{x} \odot [\![r]\!]^{(i)}) \mod 3$$
where $\odot$ denotes the Hadamard (component-wise) product modulo 3. Here, addition is also done over $\mathbb{Z}_3$.
Correctness. To see why this works, suppose that $\hat{x} \in \mathbb{Z}_2^l$. Consider any position $j \in [l]$, and denote by using a subscript j, the j^{th} position in a vector. Note that now, the position j of the output can be written as:
$$[\![x^*]\!]^{(i)}_j = [\![\hat{x}]\!]^{(i)}_j + [\![r]\!]^{(i)}_j + (\hat{x}\, [\![r]\!]^{(i)}_j \bmod 3) \mod 3$$
Consider two cases:
  - If $\hat{x}_j = 0$, then $\tilde{x}_j = x_j$. Therefore, $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]^{(i)}_j = 0 + \tilde{x}_j = x_j$.
  - If $\hat{x}_j = 1$, then $x_j = 1 - \tilde{x}_j$. Therefore, $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]^{(i)}_j = 1 + 2\tilde{x}_j \bmod 3$. If $\tilde{x}_j = 0$, this evaluates to $1 = x_j$, while if $\tilde{x}_j = 1$, it evaluates to $0 = 1 - \tilde{x}_j = x_j$

  In other words, in all cases, each component of the sum (mod 3) of shares $[\![x^*]\!]^{(i)}$ is the same as the corresponding component of x. Therefore,
$$\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]^{(i)} \,(\mathrm{mod}\ 3) = x$$
will hold.

$\mathbb{Z}_3 \to \mathbb{Z}_2$ conversion protocol $\boldsymbol{\pi}^{(3,2)}_{\mathsf{Convert}}$.

- **Functionality**: Abstractly, the goal of the protocol is to convert a sharing of x over $\mathbb{Z}_3$ to a sharing of $x^* = x \bmod 2$ over $\mathbb{Z}_2$. For our purpose, the parties will be provided with the masked input $\hat{x} = x + \tilde{x} \bmod 3$ directly, along with correlated randomness over $\mathbb{Z}_3$ (see below).
- **Preprocessing**: Each party is also given shares (over $\mathbb{Z}_2$) of two vectors: $u = \tilde{x} \bmod 2$ and $v = (\tilde{x} + \mathbf{1} \bmod 3) \bmod 2$ as correlated randomness.
- **Protocol details**: For the protocol $\boldsymbol{\pi}^{(3,2)}_{\mathsf{Convert}}(\hat{x} \mid u, v)$, each party computes its share of x^* as follows: For each position $j \in [l]$,
$$[\![x^*]\!]^{(i)}_j = \begin{cases} 1 - [\![u]\!]^{(i)}_j - [\![v]\!]^{(i)}_j & \text{if } \hat{x}_j = 0 \\ [\![v]\!]^{(i)}_j & \text{if } \hat{x}_j = 1 \\ [\![u]\!]^{(i)}_j & \text{if } \hat{x}_j = 2 \end{cases}$$
Correctness. To see why this works, consider three cases:

- If $\hat{x}_j = 0$, then $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]_j^{(i)} \bmod 2 = 1 - u_j - v_j$. This evaluates to 1 only when $\tilde{x}_j = 2$, and is exactly the case when x_j is also 1.
- $\hat{x}_j = 1$, then $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]_j^{(i)} \bmod 2 = v_j = (\tilde{x}_j + 1 \bmod 3) \bmod 2)$. This evaluates to 1 only when $\tilde{x}_j = 0$, and is exactly the case when x_j is also 1.
- $\hat{x}_j = 2$, then $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]_j^{(i)} \bmod 2 = u_j$. This evaluates to 1 only when $\tilde{x}_j = 1$, and is exactly the case when x_j is also 1.

Consequently, $\sum_{\mathrm{P}_i \in \mathcal{P}} [\![x^*]\!]^{(i)} \bmod 2 = x \bmod 2$ holds.

C.2 Composing Gate Protocols

We now describe a general technique to evaluate circuit composed of the previously specified gates in a distributed fashion. We provide details for the semi-honest fully distributed setting (with preprocessing), where all inputs are secret shared between all parties initially. This can also be thought of as a toolbox for constructing efficient distributed protocols for other constructions similar to ours. While the technique will also work for other settings (e.g., OPRF, public input), the concrete communication costs will be worse than more specially designed protocols. For these settings, we will provide more efficient protocols than provided by this general technique.

Composition protocol. Consider a circuit C (Definition 4) with input space $\mathbb{G}^{\mathsf{in}} = \prod \mathbb{G}_i^{\mathsf{in}}$. To evaluate C with input $(x_1, \ldots, x_l) \in \mathbb{G}^{\mathsf{in}}$, in the fully distributed setting, all parties are given additive shares for each x_i. Now, the distributed evaluation of C proceeds as follows:

- All vertices at the same depth in C are evaluated simultaneously, starting from the source vertices that contain the inputs of the computation.
- The evaluation of a (non-source) vertex in the graph of C is done by each party running the corresponding gate protocol locally on their share of the inputs.
- For an edge (V_a, V_b), suppose that the output of V_a is used as one of the inputs of V_b. If the gate protocol corresponding to $\mathcal{G}_V$ requires this input to be masked (e.g., the bilinear gate protocol), then before evaluating V_b, each party first masks its share of the output. Now, all parties simultaneously reveal their shares to publicly reveal the masked value. The masking values are provided to the parties in the preprocessing phase. The same value also need not be masked multiple times if it is required for multiple gates.
- The required output shares of the distributed evaluation are given by the evaluation of the sink vertices in the circuit.

Communication cost. Since the gate protocols themselves are locally computable, the communication cost during a distributed evaluation of a circuit comes solely from the public reconstructions of masked values required for gate protocols. For example, before feeding the output x of a $\mathsf{Lin}_2^{\mathbf{A}}$ gate into a $\mathsf{Convert}_{(2,3)}$ gate, in the distributed evaluation, all parties will first mask their shares of x to obtain shares of $\hat{x}$. Then, the parties will exchange messages to reconstruct the $\hat{x}$ value required for $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$.

Consider N parties taking part in the distributed evaluation. To reconstruct an l-bit value $\hat{x}$ that is additively shared among the parties, one of the following can be done.

- Each party sends its share of $\hat{x}$ to each other party. Now, all parties can compute $\hat{x}$ locally. This requires only 1 online round but has a communication cost of $(N-1)l$ bits per party. Each party sends $N-1$ messages. The simplest case for this is when $n = 2$, in which case, both parties can simultaneously exchange their shares, and add the two shares locally to reconstruct $\hat{x}$. This requires 1 online round, and has a communication cost of 1 message and l bits per party.
- All parties can send their share to a designated party, say P_1, who computes $\hat{x}$ and sends it back to everyone. This requires 2 rounds and has a communication cost of $(N-1)l$ bits for P_1 and l bits each for other parties. Here, P_1 sends $N-1$ messages while all other parties send a single message.

Reducing round complexity. It is also straightforward to parallelize the communication to reduce the number of rounds. For this, suppose that we call an edge (V_a, V_b) *communication-requiring* if the output of the protocol for V_a needs to be masked before it is input into the protocol for V_b (in other words, the gate protocol for V_b requires a masked input). Now, define the communication-depth of a vertex V as the maximum number of communication-requiring edges in any path from a source vertex to V. Now, instead of evaluating vertices with the same depth simultaneously, we will evaluate vertices with the same *communication-depth* together before the next communication round. By doing so, we can reduce the total number of rounds to the maximum communication-depth.

Preprocessing cost. A naïve technique to compute the preprocessing required is to add the preprocessing for each gate in the circuit as follows:

- The Lin and Add gates are computed locally and require no preprocessing.
- Each BL_p gate requires a Beaver-style triple which provides masks for the two inputs and a multiplication of the two masks. Specifically, for $\mathsf{BL}_p(\mathbf{K}, x)$ where $\mathbf{K} \in \mathbb{Z}_p^{l_2 \times l_1}$ and $x \in \mathbb{Z}_p^{l_1}$, the preprocessed shares are of the form $(\tilde{\mathbf{K}}, \tilde{x}, \tilde{\mathbf{K}}\tilde{x})$. Consequently, when $\mathbf{K}$ is circulant, a total of $(2l_1 + l_2)\log_p$ bits needs to be provided as preprocessing to each party.
- Each $\mathsf{Convert}_{(2,3)}$ gate requires shares of a random mask $\tilde{x}$ both over $\mathbb{Z}_2$ and $\mathbb{Z}_3$. In other words, for $x \in \mathbb{Z}_2^l$, it requires $l + \log_2 3 \cdot l$ bits of preprocessing per party.
- Each $\mathsf{Convert}_{(3,2)}$ gate requires shares of a random mask $\tilde{x}$ (over $\mathbb{Z}_3$) as well as $u = \tilde{x}$ and $v = (\tilde{x} + \mathbf{1} \bmod 3) \bmod 2$ (both over $\mathbb{Z}_2$). In other words, for $x \in \mathbb{Z}_3^l$, it requires $\log_2 3 \cdot l + 2l$ bits of preprocessing per party.

Compressing preprocessing required. We describe several optimizations to reduce the total cost of preprocessing. We assume here the presence of a trusted dealer to generate any correlations in the randomness. We note that some of the optimizations, while reducing the size of preprocessing, would be more complicated to generate if a dealer is not present.

- (Reducing redundant preprocessing). A straightforward optimization is to not mask the same value twice. For example, if the same value x is considered as input for both a BL gate and a $\mathsf{Convert}$ gate, the same mask $\tilde{x}$ can be used for both.
- (Masking BL gate outputs.) The standard BL gate requires preprocessing of the form $(\tilde{\mathbf{K}}, \tilde{x}, \tilde{\mathbf{K}}\tilde{x})$. However, if the output of the BL gate is then later input to a gate that requires a masked input (e.g., a $\mathsf{Convert}$ gate or even another BL gate), the BL gate can directly mask its output by providing $\tilde{\mathbf{K}}\tilde{x} + \tilde{y}$ instead. If this is done, the parties will compute a sharing of $\hat{y} = \mathbf{K}x + \tilde{y}$ using the BL gate. This means that the parties can directly exchange their shares to reconstruct $\hat{y}$ without requiring more preprocessing to mask y.
  For both the $\mathsf{Convert}_{2,3}$ and $\mathsf{Convert}_{(3,2)}$ gates, if the masked version $\hat{x}$ of the input x is already known to all parties, only $\log_2 3 \cdot l$ and $2l$ bits respectively of preprocessing are required per party.
- (Compression using a PRG). Another standard technique for compressing the size of preprocessing is to use a PRG. Intuitively, each party is given a different PRG seed by the trusted dealer which they can use locally to generate their randomness. Only a single party has its shares given by the dealer to ensure that the randomness is appropriately correlated.

C.3 Distributed Evaluation Protocols

Here, we provide details for the evaluation of our candidates in other settings.

C.3.1 Public-input setting

It is also straightforward to use the same generic technique to construct distributed protocols in the public-input setting. For keyed primitives, in public-input setting, the key is secret shared between the parties but the input is publicly known. The goal of the protocol is for the parties to compute shares of the output.

One useful optimization is that in the public-input setting, a BL gate where the input is known, essentially reduces to a linear gate where the key $\mathbf{K}$ is secret shared and the input x is publicly known.

2PC public-input protocol for $(2,3)$-wPRF. Concretely, for the evaluation of $(2,3)$-wPRF in the public-input setting, the first round from the distributed protocol can be entirely skipped. The two parties can directly compute shares of $w = \mathbf{K}x = \sum_{i\in\{1,2\}} [\![\mathbf{K}]\!]^{(i)} x$ locally. This also means that the preprocessing previously required for the BL gate that computed $\mathbf{K}x$ is no longer necessary. The rest of the evaluation can now proceed as before with both parties first using

$\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ to retrieve shares of w over $\mathbb{Z}_3$, and then using $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{B},3}$ to compute shares of the final output y.

In total, the evaluation takes a single round and a communication of m bits per party (to reconstruct $\hat{w}$). The only preprocessing required is for the $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ gate, for which each party will be given $m + \log_2 3 \cdot m$ bits. Furthermore, with PRG compression, P_2 will require no extra preprocessing and P_1 can be given only $\log_2 3 \cdot m$ bits.

C.3.2 3-party Distributed Evaluation

In this section, we provide a 3-party (semi-honest) protocol for computing the $(2,3)$-wPRF candidate that is secure against one passive corruption and does not require any preprocessing.

Functionality. Denote the servers by $\mathrm{P}_1, \mathrm{P}_2$, and P_3. We assume that the servers hold replicated additive shares of the key $\mathbf{K}$ and the input x. The key is assumed to be circulant and can be represented by $k \in \mathbb{Z}_2^n$. Concretely, let (k_1, k_2, k_3) and (x_1, x_2, x_3) be additive shares of k and x respectively. Then, each party P_i is given k_j, x_j with $j \neq i$. At the end of the protocol, P_2 and P_3 should hold y_2 and y_3 respectively such that (y_2, y_3) is a sharing of the wPRF output y.

Protocol details.

- First, the three servers compute additive shares of the linear mapping $\mathbf{K}x$ locally using their replicated shares. Note that this can be locally since for two secret shared values $a = a_1 + a_2 + a_3$ and $b = b_1 + b_2 + b_3$, their product can be computed as $ab = \sum_{1 \leq j,k \leq 3} a_j b_k$. Since each party holds two shares of a, b in a replicated sharing scheme, each term $a_j b_k$ can be computed by at least 1 party. Suppose that the share of P_i is denoted by $[\![\mathbf{K}x]\!]^{(i)}$.
- Now, P_1 samples $\tilde{w} \xleftarrow{\$} \mathbb{Z}_2^m$, $r_2 \leftarrow \mathbb{Z}_3^m$, and sets $r_3 = \tilde{w} - r_2 \bmod 3$. In other words (r_2, r_3) is a random $\mathbb{Z}_3$ sharing of $r = \tilde{w}$. P_1 sends $[\![\mathbf{K}x]\!]^{(1)} + \tilde{w}$ and r_i to $\mathrm{P}_{i \in \{2,3\}}$. At the same time, P_2 and P_3 exchange their shares of $\mathbf{K}x$. All of this can be done in one round.
- At this point, P_2 and P_3 can both compute $\hat{w} = \tilde{w} + \sum_{1 \leq i \leq 3} [\![\mathbf{K}x]\!]^{(i)}$. Now, for $i \in \{2,3\}$, P_i can locally compute $w_i^* \leftarrow \boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}(\hat{w} \mid r_i)$. Finally, P_i can locally compute its share y_i of the output by running $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{B},3}(\mathbf{B} \mid w_i^*)$.

Cost analysis. The protocol requires only 1 round, with two messages sent by P_1 and one message each sent by P_2 and P_3. P_1 sends a total of $2m + 2\log_2(3) \cdot m$ bits while the other two parties send m bits each. P_1 can also generate a reusable PRG seed and provide it to one of the parties which saves $\log_2(3) \cdot m$ bits of communication.

C.4 Oblivious Protocols

Our distributed evaluation protocols from Section 5.2 can be used directly for semi-honest *oblivious* PRF, or OPRF, evaluation in the preprocessing model. Recall that in the OPRF setting, one party P_1 (called the "server") holds the key $\mathbf{K}$ and the other party P_2 (called the "client") holds the input x. The goal of the protocol is to have the client learn the output of the PRF for key $\mathbf{K}$ and input x, while the server learns nothing. In the semi-honest setting, both parties can first use the distributed protocol to obtain shares of the PRF output. The server can then send its share to the client so that only the client learns the final output. Such an OPRF protocol would require one extra round over the corresponding distributed PRF protocol. We can however construct much better protocols whose efficiency rivals that of existing DDH-based OPRF protocols. Here, we provide two concrete efficient protocols for evaluating the $(2, 3)$-wPRF candidate (Construction 3) in the OPRF setting.

General structure. Both protocols take 3 rounds and involve 2 messages from the server to the client and 1 message from the client to the server. The first server message however, is only required when the key needs to be changed (or re-masked). We call this the key-update phase. Now, when the masked key is already known, our protocols are optimal in the sense that they require only a single message from the client followed by a single message from the server. Since OPRF applications usually involve reusing the same key for many PRF invocations, in such a *multi-input* setting, our protocols are comparable to other 2-round OPRF protocols in literature (e.g., DDH-based).

We detail the two protocols, $\boldsymbol{\pi}_1^{\mathsf{oprf}}$ and $\boldsymbol{\pi}_2^{\mathsf{oprf}}$, in Sections C.4.1 and C.4.2 respectively. In Section C.4.3, we compare our OPRF protocols to other common constructions in the literature. Later, in Section 7, we also report on our protocol implementations and compare their performance (both computation and communication) with related work. To foreshadow, a key observation is that in comparison to common OPRF protocols, our protocols are much faster to compute but require preprocessing as well as slightly more communication.

C.4.1 Oblivious PRF Protocol $\boldsymbol{\pi}_1^{\mathsf{oprf}}$

Our first OPRF protocol is in spirit similar to the distributed evaluation for the $(2, 3)$-wPRF construction. Since $\mathbf{K}$ is known to the server, and x is known to the client, both parties do not need to exchange their shares to reconstruct the masked values $\hat{\mathbf{K}}$ and $\hat{x}$; the party that holds a value can mask it locally and send it to the other party. This allows us to decouple the server's message that masks its PRF key from the rest of the evaluation. To update the key, the server can simply send $\hat{\mathbf{K}} = \mathbf{K} + \tilde{\mathbf{K}}$ to the client. Many PRF evaluations can now be done using the same $\hat{\mathbf{K}}$.

Preprocessing. The protocol requires the following preprocessed randomness. The mask $\tilde{\mathbf{K}}$ is given to the server only when the key-update phase needs to be run. For PRF evaluations, the trusted dealer samples $\tilde{w} \overset{\$}{\leftarrow} \mathbb{Z}_2^m$ and provides the server and client $\mathbb{Z}_2$ shares of $\tilde{w}$ along with $\mathbb{Z}_3$ shares of $r = \tilde{w}$. Additionally, the dealer also generates an OLE correlation pair $(\tilde{\mathbf{K}}, \tilde{v})$ and $(\tilde{x}, \hat{v})$ such that $\tilde{\mathbf{K}} \in \mathbb{Z}_2^{m \times n}$ is a random circulant matrix that is same for all correlations, $\tilde{v} \xleftarrow{\$} \mathbb{Z}_2^m$, $\tilde{x} \xleftarrow{\$} \mathbb{Z}_2^n$, and $\hat{v} = \tilde{\mathbf{K}}\tilde{x} + \tilde{v}$. The server is given $(\tilde{\mathbf{K}}, \tilde{v})$ while the client is given $(\tilde{x}, \hat{v})$. Note that we simply use OLE correlations and do not make use of an actual OLE protocol. In practice, if the key-update phase is run after every k evaluations (where k is known), the OLE correlations for all evaluations can be preprocessed at the beginning.

Protocol details. Assuming that the masked key $\hat{\mathbf{K}}$ is known to the client, for an input x, the evaluation protocol now proceeds as follows:

- The client computes $\hat{x} = x + \tilde{x}$ and $[\![\hat{w}]\!]^{(2)} = -\hat{\mathbf{K}}\tilde{x} + \hat{v} + [\![\tilde{w}]\!]^{(2)}$ and sends both $\hat{x}$ and $[\![\hat{w}]\!]^{(2)}$ to the server.
- The server first computes $[\![\hat{w}]\!]^{(1)} = \mathbf{K}\hat{x} - \tilde{v} + [\![\tilde{w}]\!]^{(1)}$, and adds to it the client's share to reconstruct $\hat{w}$. Identical to the distributed protocol, the server now runs $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ followed by $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{B},3}$ to obtain its share $[\![y]\!]^{(1)}$ of the PRF output. Finally, it sends both $\hat{w}$ and $[\![y]\!]^{(1)}$ to the client.
- The client also runs $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ followed by $\boldsymbol{\pi}_{\mathsf{Lin}}^{\mathbf{B},3}$ to obtain its share $[\![y]\!]^{(2)}$ of the PRF output. It can now use the server's share to reconstruct the PRF output y.

For evaluating a client input, $\boldsymbol{\pi}_1^{\mathsf{oprf}}$ takes 2 rounds and involves a single message in each direction. The client sends $2n$ bits while the server sends m bits and t $\mathbb{Z}_3$ elements. For our parameters ($n = m = 256, t = 81$), and with proper $\mathbb{Z}_3$ packing, this amounts to roughly 897 bits of total online communication. To update $\tilde{\mathbf{K}}$, the server sends a 256-bit message to the client.

C.4.2 Oblivious PRF Protocol $\boldsymbol{\pi}_2^{\mathsf{oprf}}$

For the second protocol, the server masks the PRF in a different way; a multiplicative mask is used instead of an additive one. For simplicity, suppose that $n = m$ and that the key $\mathbf{K}$ is a random full-rank circulant matrix. Then to mask $\mathbf{K}$, the server computes $\bar{\mathbf{K}} = \mathbf{RK}$ using a random matrix $\mathbf{R}$ that is also full-rank and circulant. $\mathbf{R}$ will be provided as preprocessing to the server, and can be reused for multiple PRF evaluations. The server will send $\bar{\mathbf{K}}$ to the client in the key-update phase. Note that since the product of two circulant matrices is also circulant, this message is only n bits. Additionally, since the product $\mathbf{RK}$ is essentially a convolution, it can be efficiently computed in $\Theta(n \log n)$ asymptotic runtime using the fast Fourier transform (FFT) algorithm.

Preprocessing. The protocol requires the following preprocessed randomness. The mask $\mathbf{R}$ is given to the server only when the key-update phase needs to be run. For PRF evaluations, similar to first protocol, the dealer samples $w \xleftarrow{\$} \mathbb{Z}_2^m$ and provides the server and client $\mathbb{Z}_2$ shares of $\tilde{w}$ along with $\mathbb{Z}_3$ shares of $r = \tilde{w}$. Additionally, the dealer gives $\tilde{u} \xleftarrow{\$} \mathbb{Z}_2^m$ to the client and $\tilde{v} = \mathbf{R}^{-1}\tilde{u} + \tilde{w}$ to the server.

Protocol details. Now, assuming that the masked key $\bar{\mathbf{K}}$ is known to the client, for an input x, the evaluation protocol now proceeds as follows:

- The client computes $\hat{u} = \bar{\mathbf{K}}x + \tilde{u}$ and sends it to the server.
- The server first computes $\mathbf{R}^{-1}\hat{u} + \tilde{v} = \mathbf{R}^{-1}(\mathbf{R}\mathbf{K}x + \tilde{u}) + (\mathbf{R}^{-1}\tilde{u} + \tilde{w}) = \hat{w} \mod 2$. Identical to the distributed protocol, the server now runs $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$ followed by $\boldsymbol{\pi}^{\mathbf{B},3}_{\mathsf{Lin}}$ to obtain its share $[\![y]\!]^{(1)}$ of the PRF output. Finally, it sends both $\hat{w}$ and $[\![y]\!]^{(1)}$ to the client.
- The client also runs $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$ followed by $\boldsymbol{\pi}^{\mathbf{B},3}_{\mathsf{Lin}}$ to obtain its share $[\![y]\!]^{(2)}$ of the PRF output. It can now use the server's share to reconstruct the PRF output y.

For evaluating a client input, $\boldsymbol{\pi}^{\mathsf{oprf}}_{2}$ also takes 2 rounds and involves a single message in each direction. The client sends n bits while the server sends m bits and t $\mathbb{Z}_3$ elements. This is n fewer bits of communication as compared to the first protocol. The key-update phase is slower however, since it involves a convolution rather than a simple vector addition. For our parameters ($n = m = 256, t = 81$), and with proper $\mathbb{Z}_3$ packing, this amounts to roughly 641 bits of total online communication. To update $\bar{\mathbf{K}}$, the server sends a 256-bit message to the client.

C.4.3 Comparison to other OPRF protocols

Here, we compare the concrete efficiency of our protocols with other protocols in literature.

DDH-based OPRFs. A simple and widely used OPRF is based on the Decision Diffie-Hellman (DDH) assumption. Abstractly, given a cyclic group $\mathbb{G}$ of prime order q, consider a PRF F defined as follows: For key $k \in \mathcal{K}$ and input $x \in \mathcal{X}$, define $\mathsf{F}(k, x) = H(x)^k$ where $H : \mathcal{X} \rightarrow \mathbb{G}$ is a hash function modeled as a random oracle. F is a secure PRF under the DDH assumption in the random oracle model [62].

The PRF F leads to a natural 2-party (semi-honest) protocol for oblivious evaluation. Concretely, suppose that the client holds x and the server holds k. To evaluate the PRF obliviously, the client initiates the interaction by first sampling a mask $r \xleftarrow{\$} \mathbb{Z}_q$ and then sending $a \leftarrow H(x)^r$ to the server. The server responds with $b \leftarrow a^k$. Finally, the client can retrieve the PRF computation as $y \leftarrow b^{r^{-1}}$ where r^{-1} is the multiplicative inverse of r in $\mathbb{Z}_q$. Security of this OPRF protocol has been shown in [49, 51], assuming the one-more discrete-log assumption [10] (and in the random oracle model).

We instantiate a DDH-based OPRF over the Curve25519 elliptic curve and compare its efficiency to our OPRF constructions. The key takeaway we found was that for both our constructions, the total computation time is smaller than the time it takes for a *single* elliptic curve scalar multiplication. One caveat is that our protocols require preprocessing, as well as slightly higher communication.

We point out that both of our OPRF protocols are much faster than the DDH-based OPRF although they require higher communication as well as preprocessing. To better represent the tradeoffs, we compute the minimum network speed for which the total time for one evaluation of our protocol is smaller than that for the DDH-based OPRF. We compare only the online costs for the evaluation. Analytically, we can compute that our two protocols are faster than the DDH-based OPRF protocol when the network speed is greater than ≈ 5.7Mbps and ≈ 1.85Mbps respectively. On a 50Mbps network, The communication of OPRF Protocol 1 outpaces its respective computation. The same happens with OPRF protocol 2 on an 40 Mbps network. While these numbers only present a simplified picture, we note that they should depict the overall tradeoffs for our OPRF protocols.

Other OPRFs. Many prior OPRF constructions [41, 50] require expensive exponentiations because they are based on algebraic PRFs. This means that we can expect similar performance tradeoffs for our protocols when compared to them (i.e., much faster computation, slightly more communication). [56] provides an efficient batched-OPRF protocol based on OT extension in the preprocessing. While we did not perform with the same setup, based on their performance results, we found that our protocol is substantially more efficient for a single (or a small number of) evaluation, but becomes more comparable in performance as the batch-size increases which is unsurprising considering our protocols are not optimized for the batched evaluation. Recent work [67] constructs an OPRF protocol from the Legendre PRF [36]. For 128-bit security, their protocol has a communication cost of ≈ 13KB which is substantially higher than ours.

C.5 Generating Correlated Randomness

We provide the remaining details for the randomness generation here

Bilinear correlations To compute a secure multiplication $\mathbf{K}x$ where $\mathbf{K}$ and x are both secret shared, our bilinear gate protocol requires Beaver-style preprocessed shares of the form $(\llbracket \tilde{\mathbf{K}} \rrbracket, \llbracket \tilde{x} \rrbracket, \llbracket \tilde{\mathbf{K}}\tilde{x} \rrbracket)$. These triples can be compressed using variants of existing PCGs for VOLE / OLE as shown by recent work [22, 23]. Using these techniques, n correlations can be generated with $\ll n$ bits of communication.

$(2, 3)$-correlations. We previously showed that an OT correlation can locally be converted to a $(2, 3)$-correlation when $z_0 \neq z_1$. Since P_1 knows these values, it still needs to communicate to P_2 whether to use a given correlation or not. For this, P_1 sends a single message which describes the set of instances to consider. Specifically, for a sequence of k OT-correlations, P_1 can send a k bit message where each bit signals whether to use the corresponding correlations.

Therefore, to the obtain l single-element $(2, 3)$ correlations necessary for an $x \in \mathbb{Z}_2^l$, on expectation, $1.5l$ OT correlations will be required. Naïvely, this would

OWF Params	KKW params	Sig. size (KB)
(n, m, t)	(N, M, τ)	
$(128, 453, 81)$	$(16, 150, 51)$	13.30
	$(16, 168, 45)$	12.48
	$(16, 250, 36)$	**11.54**
Picnic3-L1	$(16, 250, 36)$	12.60
$(128, 453, 81)$	$(64, 151, 45)$	13.59
	$(64, 209, 34)$	11.70
	$(64, 343, 27)$	**10.66**
Picnic2-L1	$(64, 343, 27)$	12.36

OWF Params	KKW params	Sig. size (KB)
(n, m, t)	(N, M, τ)	
$(256, 906, 162)$	$(16, 324, 92)$	50.19
	$(16, 400, 79)$	47.08
	$(16, 604, 68)$	**45.82**
Picnic3-L5	$(16, 604, 68)$	48.72
$(256, 906, 162)$	$(64, 322, 82)$	51.23
	$(64, 518, 60)$	44.04
	$(64, 604, 57)$	**43.45**
Picnic2-L5	$(64, 604, 58)$	46.18

Table 6: Signature size estimates for Picnic using $(2, 3)$-OWF, compared to Picnic using LowMC. The left table shows security level L1 (128 bits) with $N = 16$ and $N = 64$ parties, and the right table shows level L5 (256 bits).

require a $1.5l$ length message from P_1 to P_2. This can be easily compressed however, using the binary entropy function $\mathsf{H}_b(p)$ which computes the entropy of a Bernoulli process with probability p. Specifically, since we expect to throw away $p = 1/3$ fraction of the OT correlations, the message from P_1 to P_2 can be compressed to only $1.5l \cdot \mathsf{H}_b(1/3) \approx 1.377l$ bits. Consequently, on expectation, a single $(2, 3)$ correlation can be generated using just 1.377 bits of communication.

As another upshot, this means that the required $(2, 3)$ correlations can be generated on the fly even during the online phase. For example, in the $(2, 3)$-wPRF protocol, the $(2, 3)$ correlations required can be generated along with the first round of the online protocol without needing them to be given earlier as preprocessing. Note that the number of communication rounds still stays the same.

Lemma 1. *Protocol 5 securely generates a random* $(2, 3)$*-correlation.*

Proof. It is easy to see that the secret $w = w_1 + w_2 \bmod 2$ is hidden. Regardless of what $w_2 = c$ is, w_1 could be either 0 or 1 with equal probability. It is also straightforward to verify that the generated $(2, 3)$-correlation is random by iterating through all the possible cases.

$(3, 2)$-correlations.

Lemma 2. *Protocol 6 securely generates a random* $(3, 2)$*-correlation.*

Proof. Since P_1 generates $\tilde{x}_1, u_1, v_1$ randomly, $[s_0, s_1, s_2] + [(u_1 \parallel v_1), (u_1 \parallel v_1), (u_1 \parallel v_1)]$ is a random cyclic rotation of ["01", "10", "00"] where the shift depends on $\tilde{x}_1$. Now, since P_1 masks the s_j, its message to P_2 simulates an OT protocol given an OT correlation. Consequently, the security of the OT protocol will directly imply that the message from P_1 to P_2 hides $\tilde{x}_1$. The proof that the generated correlation is random is also straightforward by iterating through all possible cases.

D Signature Scheme Details

In this section we provide additional details on how to construct a signature scheme from our new primitives, in particular the $(2,3)$-OWF.

An N-party protocol. There will be N parties for our MPC protocol, each holding a secret share of x, who jointly compute $y = \mathsf{F}(x)$. The protocol tolerates up to $N-1$ corruptions: given the views of $N-1$ parties we can simulate the remaining party's view, to prove that the $N-1$ parties have no information about the remaining party's share.

The preprocessing phase is similar to that in Picnic. Each party has a random tape that they can use to sample a secret sharing of a uniformly random value (e.g., a scalar, vector, or a matrix with terms in $\mathbb{Z}_2$ or $\mathbb{Z}_3$). Each party samples their share $[\![r]\!]$ and the shared value is implicitly defined as $r = \sum_{i=1}^{N} [\![r]\!]^{(i)}$.

We must also be able to create a sharing mod 3, of a secret shared value mod 2. Let $\tilde{w} \in \mathbb{Z}_2$ be secret shared. Then to establish shares of $r = \tilde{w} \pmod 3$, the first $N-1$ parties sample a share $[\![r]\!]$ from their random tapes. The N-th party's share is chosen by the prover, so that the sum of the shares is r. We refer to the last party's share as an *auxiliary value*, since it's provided by the prover as part of pre-processing. For efficiency, the random tape for party i is generated by a random seed, denoted seed_i, using a PRG. The state of the first $N-1$ parties after pre-processing is a seed value used to generate the random tape, and for the N-th party the state is the seed value plus the list of auxiliary values, denoted aux.

After pre-processing, the parties enter the *online* phase of the protocol. The prover computes $\hat{x} = x + \tilde{x}$, where $\tilde{x}$ is a random value, established during preprocessing so that each party has a share $[\![\tilde{x}]\!]$. The parties can then compute the OWF using the homomorphic properties of the secret sharing, and the share conversion gate (to convert shares mod 2 to mod 3, used when computing z) setup during preprocessing that we describe below. During the online phase, parties broadcast values to all other parties and we write msgs_i to denote the broadcast messages of party i.

Preprocessing phase. Preprocessing establishes random seeds of all parties and shares of

1. $\tilde{x}$: a random vector in $\mathbb{Z}_2^n$, //Sampled from random tapes
2. $\tilde{w}$: the vector $\mathbf{A}\tilde{x}$ in $\mathbb{Z}_2^m$,
3. r: a sharing of $\tilde{w} \mod 3$, shares in $\mathbb{Z}_3^m$, //Tapes + one aux value
4. $\overline{r}$: a sharing of $1 - \tilde{w} \mod 3$, shares in $\mathbb{Z}_3^m$. //Computed from $[\![r]\!]$

The shares of $\overline{r}$ are computed from shares of r as follows (all arithmetic in $\mathbb{Z}_3^m$): the first party computes $[\![\overline{r}]\!] = 1 - [\![r]\!]$, then the remaining parties compute $[\![\overline{r}]\!] = -[\![r]\!]$. Then observe that

$$\sum_{i=1}^{N} [\![\overline{r}]\!]^{(i)} = 1 - [\![r]\!]^{(1)} - \ldots - [\![r]\!]^{(N)} = 1 - \sum_{i=1}^{N} [\![r]\!]^{(i)} = 1 - r$$

as required.

Online phase. The public input to the online phase is $\hat{x} = x + \tilde{x}$.

1. The parties locally compute $\hat{w} \in \mathbb{Z}_2^m$ as $\hat{w} = \mathbf{A}\hat{x}$ (since both $\hat{x}$ and $\mathbf{A}$ are public).
2. Let z be a vector in $\mathbb{Z}_3^m$ and let z_i denote the i-th component. Each party defines

$$[\![z_i]\!] = \begin{cases} [\![r_i]\!] & \text{if } \hat{w}_i = 0 \qquad \text{//Note that } [\![r_i]\!] = [\![w'_i]\!] \\ [\![\overline{r}_i]\!] & \text{if } \hat{w}_i = 1 \qquad \text{//Note that } [\![\overline{r}_i]\!] = [\![1 - w'_i]\!] \end{cases}$$

   then locally computes $[\![y]\!] = \mathbf{B}\,[\![z]\!]$. All parties broadcast $[\![y]\!]$ and reconstruct the output $y \in \mathbb{Z}_3^t$. In this step each party broadcasts t values in $\mathbb{Z}_3$.

Correctness. The protocol correctly computes the $(2, 3)$-OWF. The first step computes $w = \mathbf{A}x$, updating the public value $\hat{x} = x + x'$ with $\hat{w} = w + \tilde{w}$. The second step is where the bits of w are cast from $\mathbb{Z}_2$ to $\mathbb{Z}_3$. The parties have sharings of $\tilde{w}$ and $1 - \tilde{w} \bmod 3$ (we focus on a single bit here, for simplicity). The key observation is that when $\hat{w} = 0$, then w and $\tilde{w}$ are the same, and when $\hat{w} = 1$, w and $\tilde{w}$ are different. So in the first case we set the shares of $z = w \bmod 3$ to the shares of $[\tilde{w}] \bmod 3$, and when $\hat{w} = 1$, we set the shares of z to the complement of $\tilde{w}$.

Communication costs. Here we quantify the cost of communication for the MPC inputs, the aux values and the broadcast msgs of one party, as this will directly contribute to the signature size in the following section. Let ℓ_3 be the bit length of an element in $\mathbb{Z}_3$; the direct encoding has $\ell_3 = 2$, but with compression we can reduce ℓ_3 to as little as $\log_2(3) \approx 1.58$. [5] The size of the aux information is $m\ell_3$, the MPC input value has size n bits, and the broadcast values have size $t\ell_3$ bits (per party). The total in bits is thus

$$|\mathsf{MPC}(n, m, t)| = m\ell_3 + n + t\ell_3 \ . \qquad (5)$$

For the parameters $(n, m, t) = (128, 453, 81)$ the total is 972 bits (L1 security: 128-bit classical, 64-bit quantum) and when $(n, m, t) = (256, 906, 162)$ the total is 1943 bits (L5 security: 256-bit classical, 128-bit quantum). This compares favorably to Picnic at the same security level, which communicates 1161–1328 bits at L1 and 2295–2536 bits at L5, depending on whether LowMC uses a full or partial S-box layer [53].

Signature scheme Given the MPC protocol above, we can compute the values $\hat{x}$, aux and msgs for the $(2, 3)$-OWF and neatly drop it into the KKW proof system used in Picnic. The signature generation and verification algorithms for the $(2, 3)$-OWF signature scheme are given in Fig. 4.

We use a cryptographic hash function $\mathsf{H} : \{0, 1\}^* \to \{0, 1\}^{2\kappa}$ for computing commitments, and the function Expand takes as input a random 2κ-bit string

[5] To compress a vector $v \in \mathbb{Z}_3^n$, convert it to the integer it represents: $V = \sum_{i=0}^{n} v_i^i$ and output the binary representation of V.

and derives a challenge having the form $(\mathcal{C}, \mathcal{P})$ where $\mathcal{C}$ is a subset of $[M]$ of size τ, and $\mathcal{P}$ is a list of length τ, with entries in $[N]$. The challenge $(\mathcal{C}, \mathcal{P})$ defines τ pairs (c, p_c) where c is the index of an MPC instance for which the verifier will check the online phase, and p_c is the index of the party that will remain unopened.

Optimizations and simplifications. For ease of presentation, Fig. 4 omits some optimizations that are essential for efficiency, but are not unique to the $(2,3)$-signature schemes, they are exactly as in Picnic. All random seeds in a signature are derived from a single random root seed, using a binary tree construction. First we derive M initial seeds, once for each MPC instance, then from from the initial seed we derive the N per-party seeds. This allows the signer to reveal the seeds of $N-1$ parties by revealing only $\log_2(N)$ intermediate seeds, similarly, the initial seeds for $M-\tau$ of M instances may be revealed by communicating only $(\tau)\log_2(M/\tau)$ κ-bit seeds.

For the commitments $h'^{(k)}$ to the online execution, τ are recomputed by the verifier, and the prover provides the missing $M-\tau$. Here we compute the $h'^{(k)}$ as the leaves of a Merkle tree, so that the prover can provide the missing commitments by sending only $\tau\log_2(M/\tau)$ 2κ-bit digests.

Finally, Fig. 4 omits a random salt, included in each signature, as well as counter inputs to the hash functions to prevent multi-target attacks [37]. Also, hashing the public key when computing the challenge, and prefixing the inputs to H in each use for domain separation should also be done, as in [68].

Signature size The size of the signature in bits is:

$$\underbrace{\kappa\tau\log_2\left(\frac{M}{\tau}\right)}_{\text{initial seeds}} + \underbrace{2\kappa\tau\log_2\left(\frac{M}{\tau}\right)}_{\text{Merkle tree commitments}} + \tau\left(\underbrace{\kappa\log_2 N}_{\text{per-party seeds}} + \underbrace{|\mathsf{MPC}(n,m,t)|}_{\text{one MPC instance, Eq. (5)}}\right)$$

and we note that the direct contribution of OWF choice is limited to $|\mathsf{MPC}(n,m,t)|$[6]. However, the size of this term can impact the choice of (N, M, τ). The Picnic parameters (N, M, τ) must be chosen so that the soundness error,

$$\epsilon(N, M, \tau) = \max_{M-\tau\leq k\leq M}\left\{\frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau}N^{k-M+\tau}}\right\}.$$

is less than $2^{-\kappa}$. By searching the parameter space for fixed N and various options for M, τ, we get a curve, and choose from the combinations in the "sweet spot", near the bend of the curve with moderate computation costs. This part of the curve is similar as in Picnic, and we present some options from it in **??**.

[6] The size $|\mathsf{MPC}(n,m,t)|$ is a slight overestimate since for $1/N$ instances we don't have to send aux, if the last party is unopened. In Table 6 our estimates include this, but it's a very small difference as τ is quite small.

$(2,3)$-OWF Signatures

Inputs Both signer and verifier have F, $y = \mathsf{F}(x)$, the message to be signed Msg, and the signer has the secret key x. The parameters of the protocol (M, N, τ) are described in the text.

Commit For each MPC instance $k \in [M]$, the signer does the following.

1. Choose uniform $\mathsf{seed}^{(k)}$ and use to generate values $(\mathsf{seed}_i^{(k)})_{i \in [N]}$, and compute $\mathsf{aux}^{(k)}$ as described in the text. For $i = 1, \dots N-1$, let $\mathsf{state}_i^{(k)} = \mathsf{seed}_i^{(k)}$ and let $\mathsf{state}_N^{(k)} = \mathsf{seed}_N^{(k)} || \mathsf{aux}^{(k)}$.
2. Commit to the preprocessing phase:
$$\mathsf{com}_i^{(k)} = \mathsf{H}(\mathsf{state}_i^{(k)}) \text{ for all } i \in [N], \quad h^{(k)} = \mathsf{H}(\mathsf{com}_1^{(k)}, \dots, \mathsf{com}_N^{(k)}).$$
3. Compute MPC input $\hat{x}^{(k)} = x + \tilde{x}^{(k)}$ based on the secret key x and the random values $\tilde{x}^{(k)}$ defined by preprocessing.
4. Simulate the online phase of the MPC protocol, producing $(\mathsf{msgs}_i^{(k)})_{i \in [N]}$.
5. Commit to the online phase: $h'^{(k)} = \mathsf{H}(\hat{x}^{(k)}, \mathsf{msgs}_1^{(k)}, \dots, \mathsf{msgs}_N^{(k)})$.

Challenge The signer computes $\mathsf{ch} = \mathsf{H}(h_1, \dots h_M, h'_1, \dots, h'_M, \mathsf{Msg})$, then expands ch to the challenge $(\mathcal{C}, \mathcal{P}) := \mathsf{Expand}(\mathsf{ch})$, as described in the text.

Signature output The signature σ on Msg is
$$\sigma = (\mathsf{ch}, ((\mathsf{seed}^{(k)}, h^{(k)})_{k \notin \mathcal{C}}, (\mathsf{com}_{p_k}^{(k)}, (\mathsf{state}_i^{(k)})_{i \neq p_k}, \hat{x}^{(k)}, \mathsf{msgs}_{p_k}^{(k)})_{k \in \mathcal{C}})_{k \in [M]})$$

Verification The verifier parses σ as above, and does the following.

1. Check the preprocessing phase. For each $k \in [M]$:
   (a) If $k \in \mathcal{C}$: for all $i \in [N]$ such that $i \neq p_k$, the verifier uses $\mathsf{state}_i^{(k)}$ to compute $\mathsf{com}_i^{(k)}$ as the signer did, then computes $h'^{(k)} = \mathsf{H}(\mathsf{com}_1^{(k)}, \dots, \mathsf{com}_N^{(k)})$ using the value $\mathsf{com}_{p_k}^{(k)}$ from σ.
   (b) If $k \notin \mathcal{C}$: the verifier uses $\mathsf{seed}^{(k)}$ to compute $h'^{(k)}$ as the signer did.
2. Check the online phase:
   (a) For each $k \in \mathcal{C}$ the verifier simulates the online phase using $(\mathsf{state}_i^{(k)})_{i \neq p_k}$, masked witness $\hat{x}$ and $\mathsf{msgs}_{p_k}^{(k)}$ to compute $(\mathsf{msgs}_i)_{i \neq p_k}$. Then compute $h^{(k)}$ as the signer did. The verifier outputs 'invalid' if the output of the MPC simulation is not equal to y.
3. The verifier computes $\mathsf{ch}' = \mathsf{H}(h_1, \dots h_M, h'_1, \dots, h'_M, \mathsf{Msg})$ and outputs 'valid' if $\mathsf{ch}' = \mathsf{ch}$ and 'invalid' otherwise.

Fig. 4: Picnic-like signature scheme using the $(2,3)$-OWF and the KKW proof system.

E Deferred Implementation Details

Here, we provide the details for our optimization techniques.

Bit packing for $\mathbb{Z}_2$ vectors. Instead of representing each element in a $\mathbb{Z}_2$ vector separately, we pack several elements into a machine word and operate on them together in an SIMD manner. For our architecture with 64-bit machine words, we can pack a vector in $\mathbb{Z}_2^{256}$ (e.g., the input x) into 4 words. Since the key $\mathbf{K}$ is circulant and can be represented with $n = 256$ bits, it can also be represented by 4 words. This results in a theoretical $\times 64$ maximum speedup in run-time for operations involving $\mathbf{K}$ and x.

Bit slicing for $\mathbb{Z}_3$ vectors. We represent each element in $\mathbb{Z}_3$ using the two bits from its binary representation. For $z \in \mathbb{Z}_3$, the two bits are the least significant bit (LSB) $l_z = z \bmod 2$, and the most significant bit (MSB) h_z which is 1 if $z = 2$ and 0 otherwise. $\mathbb{Z}_3$ vectors are now also represented by two binary vectors, one containing the MSBs, and one containing the LSBs. Operations involving a $\mathbb{Z}_3$ vector are translated to operations on these binary vectors instead. We also take advantage of the bit packing optimization when operating on the binary vectors. In Table 7, we specify how we perform common operations on $\mathbb{Z}_3$ elements using our bit slicing approach.

Operation	Result MSB	Result LSB
$z_1 + z_2 \bmod 3$	$(l_1 \vee l_2) \oplus (l_1 \vee h_2) \oplus (l_2 \vee h_1)$	$(h_1 \vee h_2) \oplus (l_1 \vee h_2) \oplus (l_2 \vee h_1)$
$-z_1 \bmod 3$	l_1	h_1
$z_1 z_2 \bmod 3$	$(l_1 \wedge l_2) \oplus (h_1 \wedge h_2)$	$(l_1 \wedge h_2) \oplus (h_1 \wedge l_2)$
$\mathrm{MUX}(s; z_1, z_2)$	$(h_2 \wedge s) \vee (h_1 \wedge \neg s)$	$(l_2 \wedge s) \vee (l_1 \wedge \neg s)$

Table 7: Operations in $\mathbb{Z}_3$. z_1 and z_2 are elements in $\mathbb{Z}_3$ with (MSB, LSB) $=$ (h_1, l_1) and (h_2, l_2) respectively. For a bit s, the operation $\mathrm{MUX}(s; z_1, z_2)$ outputs z_1 when $s = 0$ and z_2 when $s = 1$.

Lookup table for matrix multiplication. Recall that the $(2,3)$-wPRF evaluation contains a mod-3 linear map using a public matrix $\mathbf{B} \in \mathbb{Z}_3^{81 \times 256}$. Specifically, it computes the matrix-vector product $\mathbf{B}w$ where $w \in \mathbb{Z}_3^{256}$. Since $\mathbf{B}$ is known prior to evaluation, we can use a lookup table to speedup the multiplication by $\mathbf{B}$[6]. The same preprocessing can also by reused for multiple evaluations of the wPRF.

For this, we partition $\mathbf{B}$, which has $m = 256$ columns, into 16 slices of 16 columns each. These matrices, denoted by $\mathbf{B}_1, \ldots, \mathbf{B}_{16}$, are all in $\mathbb{Z}_3^{81 \times 16}$. Now, for each $\mathbf{B}_i$, we will effectively build a lookup table for its multiplication with any $\mathbb{Z}_3$ vector of length 16. A point to note here is that since we represent $\mathbb{Z}_3$

vectors by two binary vectors (from the bit slicing optimization), it is sufficient to preprocess multiplications (modulo 3) for binary vectors of length 16. To multiply $\mathbf{B}_i$ by a vector in $\mathbb{Z}_3^{16}$, we can first multiply it separately by the corresponding MSB and LSB vectors, and then subtract the former from the latter modulo 3. This works since for $z_1, z_2 \in \mathbb{Z}_3$ the multiplication $z_1 z_2 \bmod 3$ can be given by $z_1(2 \cdot h_2 + l_2) \bmod 3 = z_1 l_2 - z_1 h_2 \bmod 3$ where h_2, l_2 are the MSB and LSB of z_2 respectively. Now, to multiply $\mathbf{B}$ by $v \in \mathbb{Z}_3^{256}$, we first evaluate all multiplications of the form $\mathbf{B}_i v_i$ where v_i is the $\mathbb{Z}_3^{16}$ vector denoting the i^{th} slice of v if it was split into 16-element chunks. Then, multiplication by $\mathbf{B}$ is given by $\mathbf{B}v = \sum_{i \in [16]} \mathbf{B}_i v_i \mod 3$.

In general, a $\mathbf{B} \in \mathbb{Z}_3^{t \times m}$ can be partitioned into m/c partitions with c columns each (assume c divides m for simplicity), and would require a total multiplication lookup table size of $(m/c) \cdot 2^c \cdot t$ $\mathbb{Z}_3$ elements. Multiplying $\mathbf{B}$ by $v \in \mathbb{Z}_3^m$ requires $2(m/c)$ lookup table accesses (one each for MSB and LSB of v per partition of $\mathbf{B}$) and an addition of $(m/c - 1)$ $\mathbb{Z}_3^t$ vectors.

For our parameters, this results in a table size of roughly 135MB with proper $\mathbb{Z}_3$ packing. We chose $c = 16$ as a compromise between the size of the lookup table and increased computational efficiency.