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

Itai Dinur[1], Steven Goldfeder[2], Tzipora Halevi[3], Yuval Ishai[4], Mahimna Kelkar[2,5], Vivek Sharma[6], and Greg Zaverucha[7]

[1] Ben-Gurion University
[2] Cornell Tech
[3] Brooklyn College, CUNY
[4] Technion
[5] Cornell University
[6] Graduate Center, CUNY
[7] Microsoft Research

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. (PKC 2021).
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. 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 at least 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 cryptographic primitives, such one-way functions (OWFs) [53], pseudorandom generators (PRGs) [13,65] and pseudorandom functions (PRFs) [39], 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 [32,56,62] 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 thus optimized for performance metrics relevant to the single-party setting. More recent work (see [4,41,15,3,5] 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 [45,25]. While recent general techniques from the literature [14,24] 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* [66] and protocols based on *linear secret sharing* [40,10,26]. 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 scales 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 number 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 [4] 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 OWFs and PRGs [54,38,6], these typically come at the price of bigger input size and higher nonlinear size [30,63].
- *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 [38,55,2].

The alternating moduli paradigm. The above design goals may 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 conflict, a new design paradigm was recently proposed by Boneh et al. [15] and further explored by Cheon et al. [29]. 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[8] (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 $\mathbf{K}x$ 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 map over $\mathbb{Z}_3$ to obtain an output vector y over $\mathbb{Z}_3$. (When the output is a single $\mathbb{Z}_3$ element, the final compressive map is just a sum over $\mathbb{Z}_3$.)

The above mapping from x and $\mathbf{K}$ 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 $\mathbf{K}$ has a special form. The second is a *conversion* of a mod-2 vector to a mod-3 vector, which consists of small (finite-size) parallel 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

[8] 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.

$\mathbb{Z}_2$ comes from the final linear map over $\mathbb{Z}_3$. Viewing the input as a vector over $\mathbb{Z}_3$, high degree comes from the linear map 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 [29] require a very big number of samples and are quite easy to circumvent by slightly modifying the design (as suggested in [29]).

A primary motivation for the alternating moduli paradigm was its MPC-friendliness. Indeed, several MPC protocols were proposed in [15]. These protocols demonstrated 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 classes 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 [27,36,28,49], which further motivates cryptanalysis efforts.

Remaining challenges. The initial works of [15,29] 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 [25,50] and lightweight distributed key generation for function secret sharing [22].
- Are there similar candidates where the input, output, and key are all over $\mathbb{Z}_2$? This too is motivated by natural applications.
- Can the 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 [18,19,64].

1.1 Our Contributions

1.1.1 New candidate constructions

We introduce several candidate constructions for OWF, PRG, and (weak) PRF, all based on alternation between linear maps 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 map. 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] has inputs over $\mathbb{Z}_2$ but outputs over $\mathbb{Z}_3$. This is not suitable for applications in which the output should be further processed using secret sharing over $\mathbb{Z}_2$. To this end, we propose 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 map $\mathbf{B} \in \mathbb{Z}_2^{t \times m}$, 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 $\mathbf{K}$. 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 applies an additional compressive linear map (using $\mathbf{B}$). We show how this allows our candidate to resist 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 map. It follows the same structure as the LPN-wPRF, by first expanding the input to the intermediate vector w and then applying a compressive $\mathbb{Z}_2$ linear map $\mathbf{B}$. Choosing the length m of the intermediate vector to be large enough, we can ensure that the final compressive map still results in an output of size $t = 2n$.

1.1.2 Cryptanalysis and implications on parameter choices

Algebraic attacks. 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 on combinatorial attacks and statistical tests.

OWF. 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 subset-sum algorithms based on the *representation technique* [42,9,16]. Compared to a standard meet-in-the-middle approach, this attack forced us to increase the parameters by about 30%.

wPRF and PRG constructions. Our candidate constructions are related to the ones proposed in [15] and recently analyzed in [29]. The latter work describes distinguishing attacks on the constructions of [15] with asymptotically exponential (yet, concretely significant) complexity. Specifically, the attack on the $(2, 3)$-wPRF candidate of [15] exploits an interaction between the structure of the circulant matrix $\mathbf{K}$ and the choice of $\mathbf{B}$ (which is fixed to the vector $\mathbf{1}$). On the other hand, our construction uses a random choice of $\mathbf{B}$ which, as we show, is unlikely to result in such an interaction. The weakness in the "LPN-style" wPRF candidate of [15] was due to conditional correlation between the key and the output. We fix it by applying an additional compressive linear map.

It is important to emphasize that [29] analyzed constructions where the output length is $t = 1$, while our constructions use $t \gg 1$. Although longer output gives better performance, it may also degrade security. For example, at the extreme end, if $t = m$ the scheme is trivially broken in polynomial time by linear algebra, forcing $t \ll m$. Our security analysis shows that our candidate constructions resist such simple linear algebra attacks. Yet, the main part of security analysis is 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 rows of the $t \times m$ matrix $\mathbf{B}$ (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. Note that larger t results in a smaller minimal distance.

We place a concrete limit of 2^{40} on the number of samples generated by our wPRF candidates with any particular key. This reduces the probability of bad events such as collisions (where the same input to the wPRF is selected twice) and undesired interactions between the input and the structured circulant matrix $\mathbf{K}$. More details about such inputs are given in the security analysis.

Concrete parameters. 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 larger than $2s$ and result in less efficient protocols. A lower bound on m in case $n = s$ is deduced by a subset-sum attack which resembles the one on the $(2,3)$-OWF construction. Yet, optimizations that exploit the additional data available may be possible. While we do not expect security to degrade sharply in this case, we leave the concrete analysis for this parameter setting to future work. On the other hand, setting $n = 2s$ for the $(2,3)$-OWF would also require doubling the size of the output,[9] once again, degrading efficiency.

Our constructions are new and it is not unlikely that some will be broken and require updating the parameter sets (even the "conservative" ones). Conversely, if for some of our constructions the more aggressive parameter sets turn out to resist future analysis, we would gain further 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 partic-

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

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
	$(2.5s, 2.5s, s/\log 3)$	conservative
LPN-PRG	$(s, 3s, 2s)$	
LPN-wPRF	$(2s, 2s, s)$	

Table 1: Concrete parameters for s-bit security.

ularly 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

As discussed above, our design criteria are guided by the goal of supporting efficient MPC protocols for distributed evaluation. We consider semi-honest protocols in several standard MPC models, either with or without preprocessing.

Efficient protocols. For our wPRF candidates, we present protocols in several different settings: (1) 2PC with preprocessing, where the input, key, and output are all secret-shared between the parties; (2) 3PC with one passive corruption, and (3) an OPRF-style 2PC with preprocessing, where one party holds the key and the other holds the input. For the $(2,3)$-wPRF candidate, our 2PC protocols perform 1.5-5x better than the protocols from [15] for the same functionality, in both online communication and preprocessing size. For instance, in the 2PC setting, our protocol requires 2 rounds, 1536 bits of online communication, and 662 bits of preprocessing (i.e., correlated randomness). 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 and 641 bits of online communication while the one from [15] requires 4 rounds and roughly 1800 bits of online communication.

A key ingredient for the efficiency improvement is a subprotocol for modulus conversion gates that switch between shares in $\mathbb{Z}_2$ and $\mathbb{Z}_3$ using circuit-dependent correlations. While [15] used OT in their protocols, we use these modulus conversion gates for better efficiency. We note that the same blueprint can also be used to construct efficient distributed protocols for other variants of our constructions.

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 [8] 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 [18,19,59], 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 [21] 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 [43] or "silent" [19,64] 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 over the protocols 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

MPC protocols for the symmetric primitives we consider in this work are useful for a variety of cryptographic applications. Here we discuss some of these motivating applications.

Digital signatures. Using the MPC-friendliness of candidates, we can efficiently prove knowledge of an input (e.g., of an OWF input, wPRF key, or PRG seed), using proof protocols based on the MPC-in-the-head paradigm [45]. This is the approach taken by many recently designed post-quantum signature schemes [25,51,11,12,58,7], 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 optimized 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 [25,51,50,61], a third round candidate in the NIST Post-Quantum Cryptography Standardization Process.[10] We replace the OWF used in Picnic (an instance of the LowMC block cipher [4], which is assumed to be a OWF), update the MPC protocol accordingly, and quantify the resulting signature sizes. Using our conservative $(2, 3)$-OWF parameters, 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 MPC parameter choices). This shows that OWFs based on alternating moduli are competitive with block-cipher based designs, with potential for future improvements, and we can choose a OWF with an (arguably) simpler mathematical description, without sacrificing performance.

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

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

only 2 rounds and 641 bits of online communication. Compared to a standard DDH-based OPRF [46,47], which require 512 bits of communication for 128-bit security, our protocol requires slightly higher communication but has a much faster online computation, which typically forms the efficiency bottleneck. In particular, 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 [37,48] that are based on number theoretic PRFs. Note that, unlike OPRFs based on number theoretic assumptions, ours provide plausible post-quantum security. Motivated by the latter goal, recent works [41,60] construct an OPRF protocol from the Legendre PRF [31]. For 128-bit security and only a *single* output bit, the recent protocol from [60] has online communication cost of 13KB, substantially higher than ours (with 128 output bits), and with a higher computational cost.

Fully distributed wPRF. Unlike the OPRF setting, in which one party holds the PRF key and another holds the input, there are settings in which both the input and the key need to be distributed between two or more parties. In this setting, most of the techniques for efficient OPRF protocols (including the DDH-based protocols discussed above) do not apply. One motivating application for fully distributed wPRF, already considered in [44,15], is a distributed implementation of *searchable symmetric encryption* (SSE) service. In distributed SSE, a client can obtain a decryption key of database entries matching a chosen keyword w by interacting with two or more servers, while keeping the keyword w secret. To this end, the client secret-shares w between the servers, who also hold shares of a wPRF key. Following interaction between the servers, the servers reveal the wPRF output to the client. This output can be used by the client to decrypt database entries associated with keyword w.

Secret-output wPRF. 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 to one or more parties. 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, where 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 [57]. 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) [22] is a useful tool for a variety of cryptographic applications; see [17,21] 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 [35], 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 [14,24] 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], but 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 a set $\mathcal{X}$. $\mathsf{Funcs}[\mathcal{X},\mathcal{Y}]$ 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 $v\in\mathbb{G}$, we use $\tilde{v}$ to denote a random mask 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 mod p for $\mathbb{Z}_p^l$) for the summands.

We now briefly recall standard symmetric primitives.

Definition 1 (Weak Pseudorandom Function (wPRF)). *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 infinitely many $\lambda\in\mathbb{N}$ and all adversaries $\mathcal{A}$*

running in time at most $t(\lambda)$, the following holds: taking $f_\lambda \xleftarrow{\$} \mathsf{Funcs}[\mathcal{X}_\lambda, \mathcal{Y}_\lambda]$, $k \xleftarrow{\$} \mathcal{K}_\lambda$, and $x_1, \ldots, x_l \xleftarrow{\$} \mathcal{X}_\lambda$, 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 (OWF)). *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 $t(\lambda)$, we have that,*

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

Definition 3 (Pseudorandom Generator (PRG)). *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 $x_1, \ldots, x_l \xleftarrow{\$} \mathcal{X}_\lambda$ $y_1, \ldots, y_l \xleftarrow{\$} \mathcal{Y}_\lambda$, we have that,*

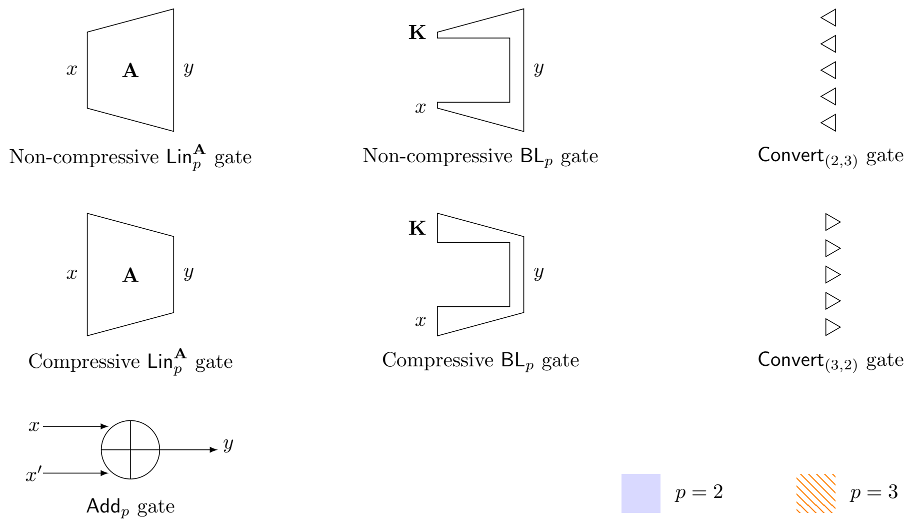
$$\left| \Pr\left[\mathcal{A}\left(1^\lambda, \{\mathsf{F}_\lambda(x_i)\}_{i\in[l]}\right)\right] - \Pr\left[\mathcal{A}\left(1^\lambda, \{y_i\}_{i\in[l]}\right)\right] \right| \leq \varepsilon(\lambda).$$

3 Candidate Constructions

In this section, we introduce our suite of candidate constructions for a number of cryptographic primitives: weak pseudorandom function families (wPRF), one-way functions (OWF), and pseudorandom generators (PRG). Our constructions are all based on alternating mod-2 and mod-3 linear maps. Given the wide range of candidates we propose, we find it useful to have a clean and unified way to describe the candidate constructions in a way that will later (in Section 5) support a unified design of matching MPC protocols.

Circuit gates. We make use of five types of basic operations, or "gates," which we detail below. All our constructions can be succinctly represented using just these gates. 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 $\mathsf{BL}_p(\mathbf{K}, x)$.



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

Fig. 1: Pictorial representations of the circuit gates. For the linear and bilinear gates, non-compressive means that the length of the output vector is greater than or equal to the length of the input vector, while compressive 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.

- $\mathbb{Z}_2 \to \mathbb{Z}_3$ **conversion.** For a positive integer l, the gate $\mathsf{Convert}^l_{(2,3)}(\cdot)$ 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 $\mathsf{Convert}^l_{(3,2)}(\cdot)$ takes as input a vector $x \in \mathbb{Z}_3^l$ and computes its map 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 $\mathsf{Convert}_{(3,2)}(x)$.

The Lin and the BL gates will behave very differently in the context of 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 $\mathbb{Z}_3$. Fig. 1 pictorially represents each circuit gate.

Construction styles. The candidate constructions we introduce follow one of two broad styles which we detail below. A wPRF construction for the first style

$(2,3)$-constructions

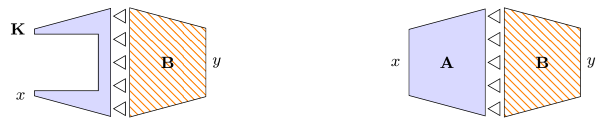
Parameters. Let λ be the security parameter and define parameters n, m, t as functions of λ such that $m \geq n, m \geq t$.
Public values. Let $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ and $\mathbf{B} \in \mathbb{Z}_3^{t \times m}$ be fixed public matrices chosen uniformly at random. The matrices can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

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 map is applied, followed by a linear mod q map. Note that after the mod p map, the input is first reinterpreted as a vector over $\mathbb{Z}_q$. For unkeyed primitives (e.g., OWF), both maps are public, while for keyed primitives (e.g., wPRF), the key is used for the first linear map. 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 maps 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 analyze this style of construction for $(p,q) = (2,3)$ and $(3,2)$ since these are arguably the simplest constructions that employ linear maps over alternate moduli. We find that the $(2,3)$-constructions outperform the $(3,2)$-

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* $y = \mathsf{Lin}_2^{\mathbf{B}}(w)$.

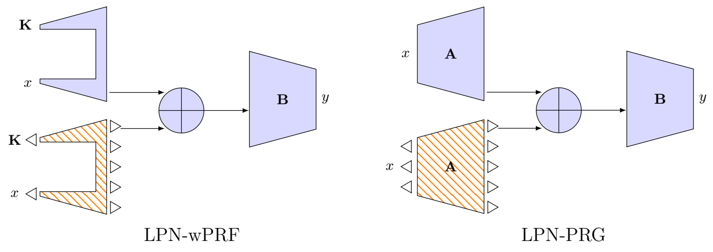


Fig. 3: LPN-style-constructions

constructions and we will primarily use the former style for our constructions. We will use $(3, 2)$-conversion gates in primitives where both the input *and the output* are shared over $\mathbb{Z}_2$.

- **LPN-style-constructions.** These constructions have the following general structure: On input x over $\mathbb{Z}_2$, first a linear $\bmod\ 2$ map given by the matrix $\mathbf{A}$ is applied to obtain u. Concurrently, the same linear map 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 map (given by $\mathbf{B}$) over $\mathbb{Z}_2$. The map $\mathbf{B}$ 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:

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

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 compressive linear map (using $\mathbf{B}$) to get the final output. This is done to resist standard attacks on LPN (see Section 4 for details).

Winning candidates. Through cryptanalysis and considering the cost for each candidate (See Sections 4 and 5 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 Figures 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 $i \in [m]$ and $j \in [n]$, 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 $m = n$ 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 (given the dimension of the matrix).

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 and defer details to the full version [34].

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.

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 $\mathbf{P}w = \mathbf{0}$. 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 $\mathbf{P}w = \mathbf{0}$ (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 $(\mathbf{0}, \hat{y}) \in \mathbb{Z}_2^{m-n} \times \mathbb{Z}_3^t$. We further note that the parity check matrix $\mathbf{P}$ defines the linear code spanned by the columns of $\mathbf{A}$. Therefore, the reduction is bi-directional, implying that inverting the $(2,3)$-OWF is equivalent to solving this special type of subset-sum problem.

Solving the subset-sum problem. We can now apply the advanced subset-sum algorithm by Howgrave-Graham and Joux [42] and the more recent ones [9,16], 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 m positive integers $(a_1, a_2, \dots, a_m)$ and a 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 [42] 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[11] of $m/2$ (which occurs 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 beats the standard meet-in-the-middle approach which requires time $2^{m/2}$.

Adaptation of previous subset-sum algorithms. The algorithm of [42] 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 [16] considers representations over $\{-1,0,1,2\}$ and we can adapt this to our setting in a similar way. In terms of complexity, ignoring polynomial factors in m, the attack of [42] runs in time $2^{0.337m}$ and uses $2^{0.256m}$ memory, while the complexity of attack of [16] 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 $(x_1, \mathbf{B}, y_1), \dots, (x_{2^r}, \mathbf{B}, y_{2^r})$ 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 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 will use at least two samples. Particularly, the subset-sum attack can also be

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

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 is 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 wPRF construction from [15] in two ways. The first transformation generates $t = 2n$ samples using a public matrix. Similarly to [15], each sample can be viewed as an LPN sample, i.e., 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 which may weaken 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-wPRF

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 MPC protocols to compute our candidate constructions in several useful 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 Section 5.4 we describe efficient protocols for distributing the generation of correlated randomness for modulus conversion gates. We defer the details of our constructions and proofs as well as protocols for other settings (3PC without preprocessing and public-input evaluation) to the full version [34].

5.1 Technical Overview

Recall that all our constructions can be succinctly represented using a set Gates of five basic gates. We will view each construction as a circuit over the basis Gates and follow the approach of [33,23] to securely evaluate such circuits using circuit-dependent correlated randomness.

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 a 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 local protocols to compute the circuit gates we use. The description of inputs (including shared correlated randomness) and outputs for each gate protocol is also summarized in Table 2. Note that the protocols work for any number of parties. 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 a generalization of Beaver's multiplication triples [8] (see, e.g., [23]). Here, we briefly provide protocols for the new modulus conversion gates.

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

- **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	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.

- **Protocol details**: For the protocol $\pi_{\mathsf{Convert}}^{(2,3)}(\hat{x} \mid r)$, each party proceeds as follows:
$$[\![x^*]\!]^{(i)} = [\![\hat{x}]\!]^{(i)} + [\![r]\!]^{(i)} + (\hat{x} \odot [\![r]\!]^{(i)}) \mod 3$$
where $\odot$ denotes the Hadamard (component-wise) product modulo 3.

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

- **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 $\pi_{\mathsf{Convert}}^{(3,2)}(\hat{x} \mid u, v)$, each party computes its share of x^* as follows: For each position $j \in [l]$, $[\![x^*]\!]_j^{(i)} = 1 - [\![u]\!]_j^{(i)} - [\![v]\!]_j^{(i)}$,$[\![v]\!]_j^{(i)}$, $[\![u]\!]_j^{(i)}$ when $\hat{x}_j = 0, 1, 2$ respectively.

In the full version, 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 $\mathsf{BL}_p^{k,l}$, $\mathsf{Convert}_{(2,3)}^{l}$, and $\mathsf{Convert}_{(3,2)}^{l}$ gates will require a preprocessing of $\log_2 p \cdot k$ bits, $\log_2 3 \cdot l$ bits, and $2l$ bits respectively.

Concrete costs. In Table 3, we provide the concrete costs for our protocols in different settings for our specific parameter choices. Preprocessing costs are based on the usage of a trusted dealer. Later, in Section 5.4, we will show how to distribute the dealer, through efficient protocols for generating the preprocessed correlations we require from standard OT-correlations. This combined with fast silent OT [20,64] makes the gap between the online cost mentioned in Table 3 and the *total* cost (including distributing the dealer) quite small. As a concrete example, the (amortized) total cost for the (2,3)-wPRF in the distributed 2PC setting is only 23% higher than the online cost when a trusted dealer is used.

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.

5.2 Distributed Evaluation in the Preprocessing Model

We briefly sketch a 2-party protocol for $(2,3)$-wPRF in the preprocessing model and defer details to the full version. 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. Note that this protocol can easily be extrapolated for distributed N-party evaluation.

5.3 Oblivious Evaluation

While our distributed protocols 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 details to the full version.

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 ($n = m = 256, t = 81$), $\boldsymbol{\pi}_1^{\mathsf{oprf}}$ has 897 bits of online communication for input evaluation. To update the key, the server sends a 256-bit message to the client.

OPRF 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. This saves 256 bits in the online phase at the expense of a slower key update phase.

5.4 Distributing the Trusted Dealer

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 $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$ protocol. The key technique we use is to convert OT correlations to the types of correlations our protocols require. Since prior work [20,19,64] 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 $\boldsymbol{\pi}_{\mathsf{Convert}}^{(2,3)}$, 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, in Protocol 5, 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*

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

- 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. 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, in Protocol 6, 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 [20,64].

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* $\tilde{x}_1 \xleftarrow{\$} \mathbb{Z}_3$, $u_1, v_1 \xleftarrow{\$} \mathbb{Z}_2$.
- *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 generating $(2,3)$-correlations 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.

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 OWF F and 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 [45]. To make the proof non-interactive, typically the Fiat-Shamir transform is used, and the message to be signed is bound to the proof by including it in the hash when computing the challenge. 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 [25], specifically the variant instantiated with the KKW proof system [51] (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 [4] 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 parts are identical to Picnic. More details can be found in the full version.

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 are full rank (using a suitable cryptographic function, such as the SHAKE extendable output function [52]). 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 $\boldsymbol{\pi}^3_{\mathsf{Add}}$, $\boldsymbol{\pi}^{\mathbf{A},2}_{\mathsf{Lin}}$, $\boldsymbol{\pi}^{\mathbf{B},3}_{\mathsf{Lin}}$, and $\boldsymbol{\pi}^{(2,3)}_{\mathsf{Convert}}$ described in Section 5, we have an N-party protocol for the $(2,3)$-

OWF Params (n,m,t)	KKW params (N,M,τ)	Sig. size (KB)
$(128,453,81)$	$(16,150,51)$	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 (n,m,t)	KKW params (N,M,τ)	Sig. size (KB)
$(256,906,162)$	$(16,324,92)$	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 4: 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).

OWF. The most challenging and costly step (in terms of communication) is the conversion gate, all other operations are done locally by the parties.

Sign and verify. 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. 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 (mostly dependent on the 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 4 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 1) both in the distributed and oblivious evaluation settings. Our implementations are in C++. For the $(2,3)$-wPRF construction, we used the

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 5: 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).

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 a roughly 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 provide benchmarks for our optimizations in Table 5.

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.

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

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

Table 6: 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.

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 6, 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 (details in the full version [34]). 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 [1] 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.

Acknowledgments. Itai Dinur is supported by ISF grants 573/16 and 1903/20, and by the European Research Council under the ERC starting grant agreement No. 757731 (LightCrypt). Yuval Ishai is supported by ERC Project NTSC (742754), ISF grant 2774/20, NSF-BSF grant 2015782, and BSF grant 2018393.

References

1. libsodium 1.0.18-stable. `https://libsodium.gitbook.io/doc/` (2020), online; December 31 2020
2. Akavia, A., Bogdanov, A., Guo, S., Kamath, A., Rosen, A.: Candidate weak pseudorandom functions in AC mod 2. In: ITCS. pp. 251–260 (2014)

3. Albrecht, M.R., Perrin, L.G.L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel structures for MPC, and more. In: ESORICS. pp. 151–171 (2019)
4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT. pp. 430–454 (2015)
5. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. TOSC **2020**(3), 1–45 (2020)
6. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. In: FOCS. pp. 166–175 (2004)
7. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: PKC. pp. 266–297 (2021)
8. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO. pp. 420–432 (1991)
9. Becker, A., Coron, J., Joux, A.: Improved generic algorithms for hard knapsacks. In: EUROCRYPT. pp. 364–385 (2011)
10. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC. pp. 1–10 (1988)
11. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: EUROCRYPT. pp. 183–211 (2020)
12. Beullens, W., de Saint Guilhem, C.: LegRoast: Efficient post-quantum signatures from the legendre PRF. In: PQCRYPTO. pp. 130–150 (2020)
13. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. SICOMP **13**(4), 850–864 (1984)
14. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: CRYPTO. pp. 67–97 (2019)
15. Boneh, D., Ishai, Y., Passelègue, A., Sahai, A., Wu, D.J.: Exploring crypto dark matter: New simple PRF candidates and their applications. In: TCC. pp. 699–729 (2018)
16. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: ASIACRYPT. pp. 633–666 (2020)
17. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: EUROCRYPT. pp. 871–900 (2021)
18. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: CCS. pp. 896–912 (2018)
19. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: CCS. pp. 291–308 (2019)
20. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO. pp. 489–518 (2019)
21. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from Ring-LPN. In: CRYPTO. pp. 387–416 (2020)
22. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT. pp. 337–367 (2015)
23. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: TCC. pp. 341–371 (2019)
24. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: CCS. pp. 869–886 (2019)

25. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: CCS. pp. 1825–1842 (2017)
26. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC. pp. 11–19 (1988)
27. Chen, L.: Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In: FOCS. pp. 1281–1304 (2019)
28. Chen, L., Ren, H.: Strong average-case lower bounds from non-trivial derandomization. In: STOC. pp. 1327–1334 (2020)
29. Cheon, J.H., Cho, W., Kim, J.H., Kim, J.: Adventures in crypto dark matter: Attacks, fixes for weak pseudorandom function candidates. In: PKC. pp. 739–760 (2021)
30. Couteau, G., Dupin, A., Méaux, P., Rossi, M., Rotella, Y.: On the concrete security of Goldreich's pseudorandom generator. In: ASIACRYPT. pp. 96–124 (2018)
31. Damgård, I.: On the randomness of legendre and jacobi sequences. In: CRYPTO. pp. 161–172 (1988)
32. Damgård, I., Keller, M.: Secure multiparty AES. In: FC. pp. 367–374 (2010)
33. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In: EUROCRYPT. pp. 167–187 (2017)
34. Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. Cryptology ePrint Archive (2021)
35. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: CCS. pp. 523–535 (2017)
36. Filmus, Y., Ishai, Y., Kaplan, A., Kindler, G.: Limits of preprocessing. In: CCC. pp. 17:1–17:22 (2020)
37. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: TCC. pp. 303–324 (2005)
38. Goldreich, O.: Candidate one-way functions based on expander graphs. In: Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation, Lecture Notes in Computer Science, vol. 6650, pp. 76–87 (2011)
39. Goldreich, O., Goldwasser, S., Micali, S.: On the cryptographic applications of random functions. In: CRYPTO. pp. 276–288 (1984)
40. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC. pp. 218–229 (1987)
41. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: CCS. p. 430–443 (2016)
42. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: EUROCRYPT. pp. 235–256 (2010)
43. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO. pp. 145–161 (2003)
44. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: CT-RSA. pp. 90–107 (2016)
45. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC. pp. 21–30 (2007)
46. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: ASIACRYPT. pp. 233–253 (2014)

47. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your Bitcoin wallet online). In: EURO S&P. pp. 276–291 (2016)
48. Jarecki, S., Liu., X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC. pp. 577–594 (2009)
49. Kabanets, V., Koroth, S., Lu, Z., Myrisiotis, D., Oliveira, I.: Algorithms and lower bounds for De Morgan formulas of low-communication leaf gates. In: CCC. pp. 15:1–15:41 (2020)
50. Kales, D., Zaverucha, G.: Improving the performance of the Picnic signature scheme. TCHES **2020**(4), 154–188 (2020)
51. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: CCS. pp. 525–537 (2018)
52. Kelsey, J., Chang, S.J., Perlner, R.: SHA-3 derived functions: cSHAKE KMAC TupleHash and ParallelHash (2016), national Institute for Standards and Technology, Special Publication 800-185.
53. Levin, L.: One-way functions and pseudorandom generators. In: STOC. pp. 363–365 (1985)
54. Matsumoto, T., Imai, H.: Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In: EUROCRYPT. pp. 419–453 (1988)
55. Miles, E., Viola, E.: Substitution-permutation networks, pseudorandom functions, and natural proofs. J. ACM **62**(6), 46:1–46:29 (2015)
56. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: ASIACRYPT. pp. 250–267 (2009)
57. Proposal, B.I.: Hierarchical deterministic wallets (2017), https://en.bitcoin.it/wiki/BIP_0032
58. de Saint Guilhem, C.D., Meyer, L.D., Orsini, E., Smart, N.P.: BBQ: Using AES in picnic signatures. In: SAC. pp. 669–692 (2019)
59. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed Vector-OLE: Improved constructions and implementation. In: CCS. pp. 1055–1072 (2019)
60. Seres, I.A., Horváth, M., Burcsi, P.: The Legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications. Cryptology ePrint Archive, Report 2021/182 (2021)
61. Team, T.P.D.: The Picnic signature algorithm specification (September 2020), version 3.0, Available at https://microsoft.github.io/Picnic/
62. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: CCS. pp. 39–56 (2017)
63. Yang, J., Guo, Q., Johansson, T., Lentmaier, M.: Revisiting the concrete security of goldreich's pseudorandom generator (2021)
64. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: CCS. pp. 1607–1626 (2020)
65. Yao, A.C.: Theory and application of trapdoor functions. In: FOCS. pp. 80–91 (1982)
66. Yao, A.C.: How to generate and exchange secrets. In: FOCS. pp. 162–167 (1986)