LaKey: Efficient Lattice-Based Distributed PRFs Enable Scalable Distributed Key Management*

Matthias Geihs
Torus Labs

Hart Montgomery
Linux Foundation

Abstract

Distributed key management (DKM) services are multi-party services that allow their users to outsource the generation, storage, and usage of cryptographic private keys, while guaranteeing that none of the involved service providers learn the private keys in the clear. This is typically achieved through distributed key generation (DKG) protocols, where the service providers generate the keys on behalf of the users in an interactive protocol, and each of the servers stores a share of each key as the result. However, with traditional DKM systems, the key material stored by each server grows linearly with the number of users.

An alternative approach to DKM is via *distributed key derivation* (DKD) where the user key shares are derived on-demand from a constant-size (in the number of users) secret-shared master key and the corresponding user's identity, which is achieved by employing a suitable distributed pseudorandom function (dPRF). However, existing suitable dPRFs require on the order of 100 interaction rounds between the servers and are therefore insufficient for settings with high network latency and where users demand real-time interaction.

To resolve the situation, we initiate the study of lattice-based distributed PRFs, with a particular focus on their application to DKD. Concretely, we show that the LWE-based PRF presented by Boneh et al. at CRYPTO'13 can be turned into a distributed PRF suitable for DKD that runs in only 8 online rounds, which is an improvement over the start-of-the-art by an order of magnitude. We further present optimizations of this basic construction. We show a new construction with improved communication efficiency proven secure under the same "standard" assumptions. Then, we present even more efficient constructions, running in as low as 5 online rounds, from non-standard, new lattice-based assumptions. We support our findings by implementing and evaluating our protocol using the MP-SPDZ framework (Keller, CCS '20). Finally, we give a formal definition of our DKD in the UC framework and prove a generic construction (for which our construction qualifies) secure in this model.

*Proceedings version. Full version available at ia.cr/2023/1254.

1 Introduction

1.1 Distributed key management

Distributed key management (DKM) systems play an important role in digital custody solutions for consumers as well as institutional customers [17, 23, 30, 40, 45, 49–51] which have an estimated market size of nearly 500 billion USD [18]. They allow a user to outsource the task of cryptographic key management to a set of servers, such that no individual server learns the key in the clear, but the key can only be accessed if sufficiently many servers cooperate. Here we are mostly concerned with m-out-of-n access structures, where there are n servers in total and to gain access to the key at least m out of the n servers need to cooperate.

At the core of a distributed key management system typically is a distributed key generation (DKG) protocol [22,31,38] that is used by the servers to generate fresh keys for onboarding users. The resulting keys are then kept in *secret-shared form* [48], which means that each server only holds some partial information about each key, and the full key can only be reconstructed by combining sufficiently many of these so-called *key shares*. The user can access its key by authenticating against the servers, downloading the key shares, and reconstructing the key locally. Alternatively, it can instruct the servers to run a secure multi-party computation (MPC) protocol [20, 29] to compute a function of the key distributedly without reconstructing the key in the clear. For example, the user may instruct the servers to run an MPC signing protocol [16, 21, 26, 36] to generate a digital signature for a given message remotely.

1.2 Scalability challenges

The security of a distributed key management system as described above relies on the assumption that a certain fraction of the servers are honest and the corresponding key shares are stored securely. To maintain security against gradual corruption, the servers must regularly run a share refresh proto-

col [37] that updates all user key shares, so that old shares can no longer be combined with new shares. A similar protocol is run if the access structure needs to be changed, e.g., for adding or removing a server. However, since key shares are stored for each user individually, the workload for all of these maintenance operations **grows linearly** with the number of users and managed keys.

1.3 Distributed key derivation

One solution to reducing the size of the key share database and the maintenance workload is to use a *distributed key derivation* (DKD) protocol instead of a standard DKG protocol for generating user keys. In contrast to a DKG, which produces a fresh and uncorrelated key on every run of the protocol, a DKD allows the servers to deterministically derive the user key on-demand from the user's identity and a constant-size secret-shared master key. Consequently, the servers no longer need to store and maintain the individual user key shares, but only need to store and maintain a single secret-shared master key. Thereby, the database size and maintenance workload are reduced from linear in the number of user keys to **constant**.

A DKD can essentially be realized by employing a distributed pseudorandom function (PRF) that satisfies a certain set of requirements. Informally speaking, a PRF is a keyed function $F_k : A \to B$ that appears indistinguishable from a randomly chosen function with the same domain and range, to any computationally-bounded observer who does not know the secret key. If the range of the PRF is equal to the key space of the targeted cryptographic scheme (e.g., an integer prime field in the case of the ECDSA signature scheme), then the PRF can be used as a key derivation function to derive a large number of user keys from a single master key and the corresponding user identities. That is, given a master PRF key k and a user identity u, the derived user key is $F_k(u)$. Note that while this user key is completely determined by k and u, yet, it appears completely random to anyone who does not know the master key k. By a *distributed PRF* we mean an interactive protocol that is run between a set of parties where the master key is secret-shared and the PRF can be evaluated without ever knowing the key in the clear.

1.4 Application requirements

For distributed key derivation, we are looking for a distributed PRF that satisfies the following properties.

User Identifiers $\subseteq$ Input(PRF). The PRF input space must contain all user identifiers, because the user keys will be derived from them.

Output(PRF) = Key Space: The PRF output distribution should be indistinguishable from the key distribution of the targeted cryptographic scheme. Specifically, in the case of ECDSA, we are looking for a uniform distribution over a prime field.

Refreshable master key shares: It should be possible to refresh the master key shares so that there is protection against gradual corruption and support for changing the access structure.

Secret-shared output: The output of the distributed PRF should be in secret-shared form so that it remains hidden from the servers and can be fed into subsequent multi-party computation protocols such as a multi-party signature generation protocol.

Robust to faulty or malicious servers: The protocol must be robust and secure against a subset of faulty or malicious servers.

Well-founded assumptions: The security of the protocol should rely on well-founded security assumptions that are reliable in real world applications.

Low round complexity: The protocol should require only a small number of interactions between the servers in order to be deployable to settings where the servers reside in different locations and network latency is high.

Why do we focus on round complexity? We focus on measuring performance in terms of round complexity because we found this to be a major bottleneck with respect to existing candidate protocols and our application. Existing distributed PRF protocols have a round complexity on the order of 100 rounds and more (cf. Table 3), where each round adds a small delay depending on the network latency, which in turn affects the user experience of the key management system.

1.5 State of the art

[34] give an overview of the state of the art of MPC-friendly PRF constructions and we mostly summarize their observations here.

Traditional PRFs (e.g., AES [28], SHA-3 [27]) are not designed for use in MPC protocols. They typically have a higher multiplicative depth than specialized constructions and work over bytes instead of prime fields.

The Naor-Reingold construction [43] yields an efficient distributed PRF protocol but has a public output and is therefore not suitable for our use case. [35] describe an efficient distributed PRF protocol based on the hardness of the Shifted Legendre Symbol Problem, which, however, is a rather unconvential and seemingly less reliable assumption [10].

LowMC [4], MiMC [1], GMiMC [3], HadesMiMC [33], and Rescue [5] are invertible block ciphers. However, invertibility is not required for our application, and a lower multiplicative complexity may be achieved by working with non-invertible functions [24]. Moreover, LowMC is defined

over binary fields and is therefore not suitable for our application where we typically work over prime fields. More specifically, there are two large issues with using LowMC: we eventually want a large uniform prime field element, and using random binary field entries to generate this is expensive. Moreover, we also use MPC that works over prime fields, so implementing LowMC in our chosen MPC framework would be very inefficient.

Farfalle [9] is an efficiently parallelizable permutation-based PRF construction with arbitrary input and output length. Ciminion [24] is a modified version of Farfalle based on a Feistel scheme. However, both require the expensive computation of a key schedule. Finally, Hydra [34] is a recent MPC-friendly PRF construction with comparable performance to Ciminion but without the need for a key schedule. Both compare favorably to the state of the art in terms of multiplicative complexity, but still require on the order of 100 online communication rounds per PRF evaluation, which incurs a significant latency overhead in our application.

Another interesting candidate for MPC-friendly PRFs is the "dark matter" PRF construction family [2,11]. However, these constructions rely on non-standard security assumptions and are therefore not ready to be used in production environments like we target with our application. In addition, the security analysis of the "dark matter" constructions focuses on outputs of vectors with small entries. Using these to generate a single large random field element is expensive, and this process might negate any efficiency advantages the construction originally had. We leave examining the security of the "dark matter" construction with larger moduli as interesting future work that could be relevant to our applications.

Another approach for distributed generation of secret shares is via Pseudo-Random Secret Sharing (PRSS) [19]. However, with this approach it is unclear how the master key can be reshared and therefore it is not suitable for a system that runs over extending periods of time (like a key management system) and protection against gradual server corruption is needed. Moreover, it only works in a setting with an honest super majority.

Other related work. [41] also addresses the problem of scalable distributed key management via on-demand distributed key derivation. In comparison to our approach, they use the lattice-based PRF construction from [12] as an almost key-homomorphic PRF, which leads to a non-interactive approximate distributed PRF protocol. Due to the PRF not being exactly key homomorphic, they have to account for a small error in the key derivation. This unfortunately makes the protocol rather complex to implement as it requires the use of particular secret sharing schemes and zero-knowledge proofs to maintain protocol efficiency, correctness, and security. In contrast, our approach does not impose any additional requirements on the used secret sharing scheme and can be directly paired with subsequent MPC protocols without the need for costly post-processing of the generated key shares.

1.6 Our contributions

We analyze the lattice-based PRF of Boneh et al. [12] and show that it can be adapted to the MPC setting to obtain a distributed PRF protocol that fulfills all our requirements (cf. Section 1.4) and runs in only $2+\log_2^2(q)+\log_2^2(q/p)$ online rounds, where q and p are lattice parameters. For concrete parameter choices derived from [13] for 128-bit security, we obtain distributed PRF protocols (REG_{12}, REG_{32}) that run in as low as 8 online rounds, which is sufficient for real-time on-demand distributed key derivation. In terms of round complexity, this is an improvement over the state-of-the-art by an order of magnitude (cf. Table 3), at the expense of increased communication. We also propose an optimized construction that leads to protocols with improved communication complexity (OPT_{12}, OPT_{32}). We support our findings by implementing and evaluating our protocols using the MP-SPDZ framework [39] in the honest majority setting with $n=3$ servers and a corruption limit of $t=1$. Finally, we model our application DKM in the UC Framework [15] and show that it can be efficiently instantiated using an MPC framework such as MP-SPDZ [39] and our distributed PRF protocol.

We envision that our DKD protocol will replace traditional key generation protocols in DKM systems that particularly target large user bases (e.g., [23,50]) as these benefit the most from an efficiency improvement per user. With our protocol, the key material of millions of users will fit within a small HSM and tasks like key share refresh can be performed by running a single instance of a share refresh protocol.

Last but not least, distributed PRFs are an important building block in MPC-based applications [34,35], and we believe that our proposed protocols may also be of interest beyond our application.

1.7 Organization

Section 2 introduces notation, basic primitives, and the lattice-based PRF construction of [12]. Section 3 describes how we adapt the construction to the MPC setting. Section 4 describes a number of enhanced constructions that further improve the efficiency. Section 5 describes the implementation of our constructions as MPC protocols and evaluates their performance. Section 6 models our application in the UC framework and shows that it can be efficiently realized using our distributed PRF protocol.

2 Preliminaries

We first introduce some basic notation. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1,\ldots,n\}$. For a random variable X we denote by $x \leftarrow X$ the process of sampling a value x according to the distribution of X. Similarly, for a finite set

S we denote by $x \leftarrow S$ the process of sampling a value x according to the uniform distribution over S. We denote by $\mathbf{x}$ a vector $(x_1, \ldots, x_{|\mathbf{x}|})$. For two bit-strings x and y we denote by $x \| y$ their concatenation. For a bit string $x \in \{0,1\}^\ell$, for every $j \in [\ell]$, let $x|_j$ denote the bit string comprising the bits j through ℓ of x. A non-negative function $f : \mathbb{N} \to \mathbb{R}$ is negligible if it vanishes faster than any inverse polynomial in some (security) parameter λ. For a group $\mathbb{G}$ of order p, element $g \in \mathbb{G}$ and a matrix $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$ (for any n and m in $\mathbb{N}$), we denote the matrix in $\mathbb{G}^{n \times m}$ whose $(i,j)^{\text{th}}$ entry is $g^{m_{i,j}}$ by $g^{\mathbf{M}}$. We denote by $\mathrm{Rk}_i(\mathbb{Z}_p^{a \times b})$ the set of all $a \times b$ matrices over $\mathbb{Z}_p$ of rank i. We use $(\mathbb{Z})$ to denote computing an operation over the integers (rather than something like $\bmod q$) when it may not be clear from context.

Distributions. By $\eta_{\mathsf{Bin}(k)}$ we denote the uniform distribution on $\{0,1\}^k$. For an $\alpha \in (0,1)$ and a prime q, the random variable $\overline{\Psi}_\alpha$ over $\mathbb{Z}_q$ is defined as $\lceil qX \rfloor \pmod q$ where X is a normal random variable with mean 0 and standard deviation $\alpha/\sqrt{2\pi}$. For two probability distributions X and Y over a finite domain D, we define their statistical distance as

$$\Delta(X,Y) = \frac{1}{2} \sum_{a \in D} |\Pr[X = a] - \Pr[Y = a]| \ .$$

We denote the uniform distribution over a finite domain D by $U(D)$. The following theorem describes a bound on the statistical distance between $U(\mathbb{Z}_m)$ and $U(\mathbb{Z}_n) \bmod m$, for $m, n \in \mathbb{N}$.

Theorem 2.1. *For $m, n \in N$, $\Delta(U(\mathbb{Z}_m), U(\mathbb{Z}_n) \bmod m) \leq m/n$.*

A similar bound is also used in [35], but left without a proof. For completeness, we include a proof of the theorem in Appendix A.1.

Rounding. We use $\lfloor \cdot \rfloor$ to denote rounding a real number to the largest integer which does not exceed it. For integers q and p where $q \geq p \geq 2$, we define the function $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor x \rfloor_p = i$, where $i \cdot \lfloor q/p \rfloor$ is the largest multiple of $\lfloor q/p \rfloor$ that does not exceed x. We also overload $\lfloor \cdot \rfloor_p$ for p a power of 2 in the natural way so that it is well-defined over $\mathbb{Z}$: in this case, we define $\lfloor x \rfloor_p$ to be eliminating the lowest $\log p$ bits of x and then outputting x/p. For a vector $\mathbf{v} \in \mathbb{Z}_q^m$, we define $\lfloor \mathbf{v} \rfloor_p$ as the vector in $\mathbb{Z}_p^m$ obtained by rounding each coordinate of the vector individually. A probability distribution χ over $\mathbb{R}$ is said to be *B-bounded* if it holds that $\Pr_{x \leftarrow \chi}[|x| > B]$ is negligible in the security parameter.

2.1 Pseudorandom Functions

A *pseudorandom function* (PRF) [32] is an efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that for a uniform k in $\mathcal{K}$ and a uniform function $f : \mathcal{X} \to \mathcal{Y}$, an oracle for $F(k, \cdot)$ is computationally indistinguishable from an oracle for $f(\cdot)$. In this paper, we allow our PRFs to be further parameterized by a public parameter pp. When needed, this pp is generated by a Setup algorithm.

Security for a PRF is defined using an experiment between a challenger and an adversary $\mathcal{A}$. For $b \in \{0,1\}$ define the following experiment $\mathsf{Expt}_b^{\mathsf{PRF}}$:

1. Given security parameter λ, the challenger samples and publishes public parameters pp to the adversary. Next, if $b = 0$ the challenger chooses a random key $k \in \mathcal{K}$ and sets $f(\cdot) := F_{\mathsf{pp}}(k, \cdot)$. If $b = 1$ the challenger chooses a random function $f : \mathcal{X} \to \mathcal{Y}$.
2. The adversary (adaptively) sends input queries $x_1, \ldots, x_Q$ in $\mathcal{X}$ and receives back $f(x_1), \ldots, f(x_Q)$.
3. Eventually the adversary outputs a bit $b' \in \{0,1\}$.

Let W_b denote the probability that $\mathcal{A}$ outputs 1 in experiment $\mathsf{Expt}_b^{\mathsf{PRF}}$.

Definition 2.2 (Pseudorandom Function)**.** A PRF $F_{\mathsf{pp}} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is secure if for all efficient adversaries $\mathcal{A}$ the quantity

$$\mathbf{Adv}^{\mathsf{PRF}}[F, \mathcal{A}] := |W_0 - W_1|$$

is negligible.

2.2 Lattice Preliminaries

Learning with errors. The *learning with errors* (LWE) problem was introduced by Regev [46] who showed that solving the LWE problem *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*. Here we define LWE with the flexibility to sample the key from a non-uniform distribution.

Definition 2.3 (Learning With Errors)**.** For integers $q = q(n) \geq 2$, a noise distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, and a key distribution Ψ over $\mathbb{Z}_q^n$, the learning with errors problem $(\mathbb{Z}_q, n, \chi, \Psi)$-LWE is to distinguish between the following pairs of distributions:

$$\{\mathbf{A}, \mathbf{A}^\intercal \mathbf{s} + \chi\} \qquad \text{and} \qquad \{\mathbf{A}, \mathbf{u}\},$$

where $m = \mathrm{poly}(n)$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \leftarrow \Psi$, $\chi \leftarrow \chi^m$, and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$. We refer to the m columns of the matrix $\mathbf{A}$ as the LWE sample points.

Regev [46] shows that for a certain noise distribution χ (a discrete Gaussian), for Ψ the uniform distribution over $\mathbb{Z}_q^n$, for n polynomial in λ, and a sufficiently large q, the LWE problem is as hard as the worst-case SIVP and GapSVP under a quantum reduction (see also [14, 44]). These results have been extended to show that Ψ can be sampled from a low norm distribution (in particular, from the noise distribution χ) and the resulting problem is as hard as the basic LWE problem [6, 14]. Similarly, the noise distribution χ can be a simple low-norm distribution [42] if m is small enough.

Learning with rounding. Banerjee, Peikert, and Rosen [7] consider a related problem, denoted the "learning with *rounding*" (LWR) problem (recall the notation $\lfloor \cdot \rceil_p$).

Definition 2.4 (Learning With Rounding)**.** For integers $q = q(\lambda)$ and $p = p(\lambda)$ such that $q > p \geq 2$ and a key distribution Ψ over $\mathbb{Z}_q^n$, the learning with rounding problem $(\mathbb{Z}_q, n, p, \Psi)$-LWR is to distinguish between the following pairs of distributions:

$$\{\mathbf{A}, \lfloor \mathbf{A}^T \mathbf{s} \rceil_p\} \qquad \text{and} \qquad \{\mathbf{A}, \lfloor \mathbf{u} \rceil_p\},$$

where $m = \text{poly}(n)$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \leftarrow \Psi$, and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$.

Banerjee et al. show that for any B-bounded distribution χ over $\mathbb{Z}$ and $q \geq pBn^{\omega(1)}$, the $(\mathbb{Z}_q, n, p)$-LWR problem is at least as hard as solving the $(\mathbb{Z}_q, n, \chi)$-LWE problem. We note that some papers in this area (e.g. [12]) define LWE and LWR to use non-uniform *samples*, and not just keys, but we do not need non-uniform samples in our proofs, so we can simplify our definitions here.

For both LWE and LWR, we sometimes treat the parameter m as an arbitrary polynomial, meaning that we can "query" and LWE/LWR "oracle" and receive any polynomial number of samples that we like. This greatly simplifies our proofs and is standard in the literature (e.g. [12]).

2.3 Lattice-Based PRF from [12]

We restate the LWE-based PRF from [12], Section 5. It will serve as the basis for our MPC-friendly PRF construction.

Construction. Let q, p, n, and m be integers such that $m = n\lceil \log q \rceil$ and p divides q. Let $\overline{\Psi}_\alpha$ be the standard LWE discrete Gaussian noise distribution with parameter α.

Let the public parameter pp be a pair of matrices of the form $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{m \times m}$ where each row of $\mathbf{A}_0$ and $\mathbf{A}_1$ is sampled from $\eta_{\text{Bin}(m)}$ such that both matrices are full rank. The secret key $\mathbf{k}$ is a vector in $\mathbb{Z}_q^m$. Define $F_{\text{LWE}} : \mathbb{Z}_q^m \times \{0,1\}^\ell \to \mathbb{Z}_p^m$ as follows:

$$F_{\text{LWE}}(\mathbf{k}, x) = \left\lfloor \prod_{i=1}^{\ell} \mathbf{A}_{x_i} \cdot \mathbf{k} \right\rceil_p .$$

Security. [12], Theorem 5.1 establishes the security of F_{LWE} under the LWE assumption. We restate the theorem here for reference.

Theorem 2.5. *The function F_{LWE} is pseudorandom under the $(\mathbb{Z}_q, n, \overline{\Psi}_\alpha)$-LWE assumption for parameter choices satisfying $\alpha \cdot m^\ell \cdot p \leq 2^{-\omega(\log n)}$.*

3 Adaptation to MPC

In the following, we describe how we adapt and optimize the lattice-based PRF construction of [12] (cf. Section 2.3) for the MPC setting, with a particular focus on the application to distributed key derivation. It turns out that highly structured, lattice-based PRFs like that of [12] compose quite efficiently with certain arithmetic MPC protocols, a fact that seemingly has not been exploited extensively in the MPC literature. The intuition is quite natural–MPC protocols designed for arithmetic circuits will quite naturally fit lattice-based schemes–but rather than explain this in detail, we will show it with performance numbers later in the paper.

We note that the PRF F_{LWE} from [12] outputs a *vector* of entries over $\mathbb{Z}_p^m$, and we need to output an integer that is uniform modulo a large prime (i.e. the ECDSA group order). At a first glance, it might be tempting to choose p to directly be this prime and output a single entry in the output vector of F_{LWE}. However, as shown in [46] (and implicitly in Theorem 2.5), the hardness of LWE (and, by corollary, LWR) is proportional to the ratio of the noise (or number of rounded bits) to the modulus. The larger p gets in our case, the worse security we have, and if we choose p to be exponentially large, there may be an efficient attack on our scheme (note that Theorem 2.5 does not hold in this case). Thus, we must compose many small LWR outputs into one larger one, which we do in our construction below.

3.1 Use a random oracle

For practical purposes, it is generally accepted to rely on the random oracle model (ROM) [8]. Relying on this model, we can replace the product $\prod_{i=1}^{\ell} \mathbf{A}_{x_i}$ by a call to a hash function $H(x)$ modeled as a random oracle. This leads to the following PRF construction.

Construction. Let q, p, l, m, and n be integers such that $m = n\lceil \log q \rceil$ and p divides q. Let $H : \{0,1\}^* \to \mathbb{Z}_q^{l \times m}$ be a cryptographic hash function that gets as input a binary string of arbitrary length and outputs a matrix in $\mathbb{Z}_q^{l \times m}$. The secret key $\mathbf{k}$ is a vector in $\mathbb{Z}_q^m$ sampled uniformly at random. Define $F_{\text{RO}} : \mathbb{Z}_q^m \times \{0,1\}^* \to \mathbb{Z}_p^l$ as follows:

$$F_{\text{RO}}(\mathbf{k}, x) = \lfloor H(x) \cdot \mathbf{k} \rceil_p .$$

Security. The following theorem establishes the security of F_{RO} under the LWE assumption in the random oracle model. We note that this follows almost immediately from the proof of hardness of learning with rounding in [7], Theorem 3.2.

Theorem 3.1. *Let n, p, and q be integers, and let χ be any B-bounded distribution over $\mathbb{Z}$ such that, for security parameter λ, $q = q(\lambda)$, $p = p(\lambda)$, $q > p \geq 2$, and $q \geq pBn^{\omega(1)}$. Any adversary that can win the PRF security game as defined in Definition 2.2 with non-negligible advantage ε can be used to solve the $(\mathbb{Z}_q, n, p)$-LWR problem with advantage ε.*

Proof. Suppose we are given access to a $(\mathbb{Z}_q, n, p)$-LWR oracle which either outputs "real" samples of the form

$\left(\mathbf{A}_i \in \mathbb{Z}_q^{l\times m}, \lfloor \mathbf{A}_i\mathbf{k}\rfloor_p \in \mathbb{Z}_p^l\right)$ for a fixed (but uniformly random) key $\mathbf{k} \in \mathbb{Z}_q^m$ and "fresh" uniformly random samples $\mathbf{A}_i$ or "random" samples $\left(\mathbf{A}_i \in \mathbb{Z}_q^{l\times m}, \mathbf{r} \in \mathbb{Z}_p^l\right)$. We show how to build a PRF simulation such that any adversary that wins the PRF security game can be used to solve the associated LWR problem.

Consider the following challenger $\mathcal{C}$ in the PRF security game. $\mathcal{C}$ will keep a database consisting of entries of the form $\left(\{0,1\}^*, \mathbb{Z}_q^{l\times m}, \mathbb{Z}_p^l\right)$. Recall that $\mathcal{C}$ has access to a $(\mathbb{Z}_q, n, p)$-LWR oracle O which is either a "real" or "random" LWR oracle. $\mathcal{C}$ does the following on respective queries:

- On query $x_i \in \{0,1\}^*$ to H, $\mathcal{C}$ checks if x_i exists in the database. If so, it responds with the *second* entry in the database (some $\mathbf{A}_i \in Z_q^{l\times m}$, as we will see soon). If not, it queries the LWR oracle, getting a query of the form $\left(\mathbf{A}_i \in \mathbb{Z}_q^{l\times m}, \mathbf{t}_i \in \mathbb{Z}_p^l\right)$, where $\mathbf{t}$ could be either "real" or "random", depending on the LWR oracle. $\mathcal{C}$ then adds the tuple $(x_i, \mathbf{A}_i, \mathbf{t}_i)$ to the database.

- On query $x_i \in \{0,1\}^*$ to F, $\mathcal{C}$ checks if x_i exists in the database. If so, it responds with the *third* entry in the database (some $\mathbf{t}_i \in Z_q^l$). If not, it queries the LWR oracle, getting a query of the form $\left(\mathbf{A}_i \in \mathbb{Z}_q^{l\times m}, \mathbf{t}_i \in \mathbb{Z}_p^l\right)$, where $\mathbf{t}$ could be either "real" or "random", depending on the LWR oracle. $\mathcal{C}$ then adds the tuple $(x_i, \mathbf{A}_i, \mathbf{t}_i)$ to the database.

Note that if the LWR oracle is "real", then $\mathcal{C}$ simulates F_{RO} perfectly. On the other hand, if the LWR oracle is "random", then $\mathcal{C}$ simulates a truly random function. Thus, any adversary that can win the PRF game with advantage ε can be used to solve the LWR problem with identical advantage, completing the proof. □

3.2 Adapt lattice parameters

Our goal is to choose the PRF parameters so that the PRF can be evaluated efficiently as an MPC protocol. In particular, we focus on minimizing the online round count, which is important for achieving low latency in our distributed key derivation application.

We observe that if we choose q and p as powers of 2, then both the modulo operation $x \mod q$ and the rounding operation $\lfloor x\rfloor_p$ can be expressed efficiently by bitwise operations. For an integer x with binary representation $x_1, \ldots, x_n$, we have

$$\lfloor x \mod q\rfloor_p = \mathsf{int}(x_{\log_2(q/p)}, \ldots, x_{\log_2(q)}) \ ,$$

where $\mathsf{int}(x_1, \ldots, x_n) = \sum_{i=1}^n x_i \cdot 2^{i-1}$.

As we will see later, this adaptation enables us to evaluate the modulo and rounding operation efficiently in the MPC setting.

3.3 Adapt message space and compose outputs

For our application, we need the PRF to output random values in a prime field $\mathbb{Z}_{p'}$, where p' is determined by the application. However, so far the lattice PRF outputs a vector in $\mathbb{Z}_p^l$, where p is a power of 2.

Adapt message space. As we would like to obtain PRF outputs in $\mathbb{Z}_{p'}$, for some prime p', we will choose $\mathbb{Z}_{p'}$ as the native message space of computation, which is also compatible with many MPC protocols. State-of-the-art MPC protocols allow for non-interactive addition and one-round multiplication over $\mathbb{Z}_{p'}$.

Compose outputs. To convert a vector of random elements over $\mathbb{Z}_p$ into a single random value over $\mathbb{Z}_{p'}$, we use the following map M_{COMP},

$$M_{\text{COMP}} : \mathbb{Z}_p^l \to \mathbb{Z}_{p'}; \mathbf{x} \mapsto \sum_{i=1}^{l} x_i \cdot p^{i-1} \ .$$

We denote the composed PRF by

$$F_{\text{COMP}} : \mathbb{Z}_q^m \times \{0,1\}^* \to \mathbb{Z}_p'; (\mathbf{k}, x) \mapsto M_{\text{COMP}}(F_{\text{RO}}(\mathbf{k}, x)) \ .$$

Security. The security of the PRF F_{COMP} follows from the security of F_{RO} (Theorem 3.1) and a bound on the statistical distance between the distribution of M_{COMP} over uniform-random inputs and the uniform distribution over $\mathbb{Z}_{p'}$ (Theorem 2.1).

Theorem 3.2. *F_{COMP} satisfies Definition 2.2.*

Proof. By a corollary of Theorem 2.1, we have that

$$\Delta\left(M_{\text{COMP}}(U(\mathbb{Z}_p^l)), U(\mathbb{Z}_{p'})\right) \le p'/p^l \ .$$

Moreover, by Theorem 3.1 we know that F_{RO} is computationally indistinguishable from $U(\mathbb{Z}_p^l)$. It follows from hybridizing the above statements that F_{COMP} is computationally indistinguishable from $U\left(\mathbb{Z}_{p'}\right)$, and that the advantage $\mathbf{Adv}^{\mathsf{PRF}}[F_{\text{COMP}}, \mathcal{A}]$ is negligible for all polynomially-bounded $\mathcal{A}$. □

4 Enhanced constructions

In this section, we show how to build an optimized version of our PRF. We gain (sometimes substantial) performance improvements at the cost of provable security. In particular, we conjecture new circular security assumptions around LWE/LWR, prove that assumptions that are "very close" are secure under the standard LWE assumption, and show how we gain performance. Due to space constraints, we are unfortunately unable to include all of our proofs in this section. For readability, we have included this full section in the appendix.

4.1 Overview of core ideas

The basic idea of our construction is to combine LWR outputs together to create some value that is computationally indistinguishable from uniform modulo some large prime p. Consider some number p_0. If we have LWR instances $t_1, ...t_\ell$ that give us some uniform value over $\mathbb{Z}_{p_0}$, then we compute the sum $\sum_{i=1}^{\ell} p_0^{i-1} t_i$. If $p_0^t - p$ is superpolynomially large, then we have constructed a uniformly random value $\mod p$ assuming that the LWR assumption is true.

In some sense, this seems a little bit wasteful: for instance, what happens if we do not compute the rounding operation of the higher-order terms (those that are multiplied by some p_0^i for $i > 1$ in the above exposition)? If we are rounding away k bits, we could instead not round away the low order bits when computing t_2 and multiply this version of t_2, for instance, by $\frac{p_0}{2^k}$. This way t_1 would "hide" the noise bits of t_2 that we haven't rounded away and hopefully we could still have security.

A concrete example. Consider two integers p_1 and p_2. We set $P_1 = 2^{p_1}$ and $P_2 = 2^{p_2}$ and then $q = P_1^2 P_2$. We also set $P = P_1^3 P_2^2$, ignoring for a brief moment that we would eventually like P to be prime. Suppose we let $\mathbf{a} \in \mathbb{Z}_q^n$ be sampled uniformly at random for some integer n and also let $\mathbf{s_1}, \mathbf{s_2} \in \mathbb{Z}_q^n$ be sampled such that each entry is random in $[0, ..., P_1]$. We could compute $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} \right\rfloor_{p_2 p_1}$ and $t_2 = \mathbf{a}^T \mathbf{s_2}$ and then set $t = t_1 + P_1 P_2 t_2$.

What is going on here? the highest order bits of t_1–P_1 of them, in fact–mask the lowest P_1 bits of t_2 that we would normally round away. It perhaps surprisingly turns out that, assuming P_1 and P_2 are large enough, we can actually prove that t is pseudorandom $\mod P$ assuming LWE holds. The proof follows from a relatively natural hybrid argument.

We can also extend this idea for higher values of ℓ: in general, setting $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} \right\rfloor_{P_2 P_1}$, $t_2 = \mathbf{a}^T \mathbf{s_2}$, ... , $t_\ell = \mathbf{a}^T \mathbf{s}_\ell$ and combining in the same style as before still works in the sense that we can prove security from LWE. This allows us to reduce the work needed in our MPC.

Moving to circular security. However, the above approach is a little bit unsatisfactory. No matter how we set ℓ, we still have to compute $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} \right\rfloor_{P_2 P_1}$, which effectively lower bounds the number of rounds needed in our MPC. Since our MPC is over arithmetic circuits, there are two expensive (because they are bitwise) operations here which must be done: computing $\mod q$, and rounding. We can try to improve things in the following ways:

- First, suppose we sample the keys $\mathbf{s}_i$ from a slightly different distribution: uniform over $[0, ..., P_1/n]$ rather than uniform over $[0, ..., P_1]$. This would give us the property that $\mathbf{a s_1} < P_1 P_2 P_1^2$, simplifying our argument.
- Then, in our computation, suppose we just compute $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} (\mathbb{Z}) \right\rfloor_{P_2 P_1}$: where we compute $\mathbf{a}^T \mathbf{s}_1$ over $\mathbb{Z}$. In other words, we *never* compute a mod q operation, and instead only round away the lowest-order p_1 bits.
- Furthermore, suppose we restrict $P_2 \geq P_1$.

We can now see how to argue that $t_1 + P_1 P_2 t_2$ is uniform mod P. Note that we cannot reduce to standard LWE, but must make a circular security-style of assumption:

- Normally, letting an adversary see $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} (\mathbb{Z}) \right\rfloor_{P_2 P_1}$ instead of $t_1 = \left\lfloor \mathbf{a}^T \mathbf{s_1} \right\rfloor_{P_2 P_1}$ where we compute mod q would break security with enough samples. However, the high-order bits in t_1 that would be "eliminated" by the mod q operation are going to correspond to the integer representation between $P_1 P_2 P_1^2$ and $P_1 P_2 P_1^2 P_2$: in other words, where the "good bits" of t_2 lie.
- The bits of t_2 that we would usually round away still lie within the "good bits" of t_1 as before.

So, in this case, the "good" bits of each LWR instance effectively mask the "sensitive" bits of the other instance. While we can't prove that this construction holds under a standard LWE assumption, we can conjecture that it stands a good chance of being secure based on the previous intuition. However, such an assumption certainly would need more study before being used in a practical system. We formalize this later in this section as the "overlapping LWR assumption".

Generalizing this intuition. As before, we can continue to chain more and more terms together for larger values of P. However, in this case we know far less about the security: it could be the case, for instance, that our assumptions hold for a constant ℓ but not if ℓ is larger. We leave a formal assessment of the security of these assumptions to future work.

We also note that, in the above, we have technically not defined what to do with the "middle" terms, i.e. t_2 through $t_{\ell-1}$: do we need to still compute the $\mod q$ operation, or is it OK if we just add them in directly, only computing the rounding operation on t_1 and the $\mod q$ operation on t_ℓ? We do not know the answer to this for sure and thus define a flexible assumption.

4.2 LWR with nonuniform key distributions

In our constructions below, we will need to use LWR with a nonuniform key distribution. We note that [6] showed that LWE with key distribution drawn from the noise distribution was (almost) exactly as hard as regular LWE for certain moduli, and [14] refined this idea further, among many other contributions. Although we cannot directly make these proofs work with rounding, we can take their core ideas and apply the standard rounding reduction of [7] to show that we can

use LWR with a nonuniform (and even non-Gaussian) key distribution. Below, we prove that LWR with an appropriate nonuniform key distribution is as hard as (relatively) standard LWE.

Theorem 4.1. *Let n, p, and $q' < q$ be integers, and let χ be any B-bounded distribution over $\mathbb{Z}$ such that, for security parameter λ, $q = q(\lambda)$, $p = p(\lambda)$, $q > p \geq 2$, and $q' \geq Bn^{\omega(1)}$ and $q \geq pBn^{\omega(1)}$. Let Ψ be the distribution over $\mathbb{Z}_q^n$ that outputs each value in the vector uniformly at random between 0 and q'. It is the case that the $(\mathbb{Z}_q, n, p, \Psi)$-LWR problem is at least as hard as solving the $(\mathbb{Z}_q, n, \chi)$-LWE problem.*

Proof intuition. We first show that LWE with the key drawn from the noise distribution is as hard as standard LWE. This is a result that has been shown numerous times in a number of different contexts [6, 14], but we found it easier to prove it directly here ourselves than to adapt these other results to our context. We then show that LWE with a key uniformly sampled over $\mathbb{Z}_{q'}$ is at least as hard as LWE with key drawn from the noise distribution, similar to a result of [14]. Finally, we apply techniques from [7] to show that our LWR assumption is at least as hard as this LWE assumption with the key sampled uniformly over $\mathbb{Z}_{q'}$. For the full proof, please see Appendix B.

A corollary for uniform noise. With a very slight tweak, we can extend our line of results above to handle uniformly sampled noise instead of rounding. We state this below in the following theorem.

Theorem 4.2. *Let n and $q' < q$ be integers, and let χ be any B-bounded distribution over $\mathbb{Z}$ such that, for security parameter λ, $q = q(\lambda)$, $p = p(\lambda)$, $q > p \geq 2$, and $q' \geq Bn^{\omega(1)}$ and $q \geq pBn^{\omega(1)}$. Let ψ be a distribution over $\mathbb{Z}_q$ that outputs a value uniformly at random between 0 and q', and let Ψ be the distribution over $\mathbb{Z}_q^n$ that outputs each value in the vector uniformly at random between 0 and q'. It is the case that the $(\mathbb{Z}_q, n, \psi, \Psi)$-LWE problem is at least as hard as solving the $(\mathbb{Z}_q, n, \chi)$-LWE problem.*

We prove this theorem in Appendix B.

4.3 Defining and proving the secure instance

We next define and prove secure the optimized instance for which we have a security proof to standard lattice assumptions. This is just a general and more formally defined version of what we discussed above.

Construction. Our construction takes as setup parameters a security parameter λ and some prime P, and then includes other integer parameters n, p_1, p_2, ℓ, n, and E. We require the following from these parameters:

- We set $P_1 = 2^{p_1}$ and $P_2 = 2^{p_2}$.
- We set $q = P_1 P_2 P_1$.
- $P_1^{\ell+1} P_2^{\ell} \geq PE$.
- P_1, P_2, and E are superpolynomially large in λ.
- We set n as a lattice dimension dependent on λ.

We assume the existence of a hash function $H : \{0,1\}^k \to \mathbb{Z}_q^n$. The secret key of our PRF F_{SEC} is a matrix $\mathbf{S} \in \mathbb{Z}_q^{n \times \ell}$ sampled so that each entry is uniform in $[0, ..., P_1]$.[1] We denote $\mathbf{S}_i$ to be the ith column of $\mathbf{S}$.

We formally define our PRF $F_{\text{SEC}} : \{0,1\}^k \times \mathbb{Z}_q^{n \times \ell} \to \mathbb{Z}_p$ as follows. On input x, we:

- Set $\mathbf{a}_x \in \mathbb{Z}_q^n = H(x)$.
- Let $c_{x,i} \in \mathbb{Z}_q = \mathbf{a}_x^T \mathbf{S}_i$.
- Output the sum $\lfloor c_{x,1} \rfloor_{P_1 P_2} + P_2 c_{x,2} + P_1 P_2^2 c_{x,3} + ... + P_1^{\ell-2} P_2^{\ell-1} c_{x,\ell} \mod P$.

Intuition of the construction. What is going on here? Each of the "LWR" instances works mod $q = P_1 P_2 P_1$ and overlaps on the top and bottom (if applicable) by P_1. So the top of each fresh instance hides the "noise bits" of the instance above it. This is the goal of the construction, and how we can gain efficiency: we only have to compute one rounding operation in entire computation of the PRF. This isn't much if we are directly computing the PRF, but may save us considerable time if we compute the PRF in an MPC.

Proof of security. As we claimed earlier, we can in fact prove the above construction secure under the standard learning with rounding assumption. We note that, as shown in [7], this follows from the LWE assumption with slightly superpolynomial (in the security parameter λ) modulus.

Our core argument is relatively straightforward: we use a hybrid argument, progressively randomizing the terms $c_{x,i}$, starting with $i = 1$ until all of the terms are random. Note that the randomness of each $c_{x,1}$ term immediately follows from the LWR assumption. Once we "switch" the $c_{x,1}$ terms to random, then they mask the lowest p_1 bits of the $c_{x,2}$ terms, meaning that we can once again apply LWR. We can then continue this argument all the way up to the $c_{x,\ell}$ terms. There are some subtleties in the proof, particularly with how we deal with "overflow" bits, but the overall idea is pretty straightforward.

We state this formally below with the following theorem:

Theorem 4.3. *Let P, n, q, p_1, P_1, p_2, P_2, ℓ, n, and E be parameters as defined above. Let Ψ denote the distribution over*

[1] We could set our keys here to be uniform over $\mathbb{Z}_q$, but we choose to set them this way for efficiency reasons (i.e., lesser bitlength leads to fewer preshared random bits consumed in MPC). As shown in [6] and myriad follow-ups, we lose little security for doing so.

$\mathbb{Z}^n$ defined by sampling each entry as an integer uniformly at random in the range $[0, P_1]$ and ψ be the distribution over Z defined by sampling an integer uniformly at random in the range $[0, P_1]$. Any adversary that can win the PRF security game as defined in Definition 2.2 with non-negligible advantage ε can be used to solve the $(\mathbb{Z}_q, n, p_1 p_2, \Psi)$-LWR problem or the $(\mathbb{Z}_q, n, \psi, \Psi)$-LWE problem with advantage polynomial in ε assuming that H is a random oracle.

We prove this theorem in Appendix B. We note that by theorems 4.1 and 4.2 (proven in Appendix B), the assumptions used in this theorem are reducible from standard LWE. So therefore the hardness of this PRF follows from standard LWE. Also, the assumptions in this proof are not substantially worse in terms of parameters than just regular LWR. So it would not be irresponsible to use this scheme in a real deployment.

4.4 Instances with only conjectured security

In this section, we discuss other PRF constructions for which we cannot prove security from standard assumptions. While the lack of security proofs is obviously big drawback of these constructions, they do offer substantial efficiency advantages compared to the provably secure PRFs, especially within the context of MPC computations.

We start with a construction that is very close to our construction with provable security.

A construction with almost a security proof. As before, our construction takes as setup parameters a security parameter λ and some prime P, and then includes other integer parameters n, p_1, p_2, ℓ, n, and E. We have the same requirements for these parameters as with F_{SEC} with the additional requirement that $p_2 \geq p_1$.

We assume the existence of a hash function $H : \{0,1\}^k \to \mathbb{Z}_q^n$. The secret key of our PRF F_{ASEC} is a matrix $\mathbf{S} \in \mathbb{Z}_q^{n \times \ell}$ sampled so that each entry is uniform in $[0, ..., P_1/n]$. We denote $\mathbf{S}_i$ to be the ith column of $\mathbf{S}$.

We formally define our PRF $F_{\text{ASEC}} : \{0,1\}^k \times \mathbb{Z}_q^{n \times \ell} \to \mathbb{Z}_p$ as follows:

- Set $\mathbf{a}_x \in \mathbb{Z}_q^n = H(x)$.
- Set $c_{x,1} = \mathbf{a}_x^T \mathbf{S}_1$ over $\mathbb{Z}$.
- For $i \geq 1$, let $c_{x,i} \in \mathbb{Z}_q = \mathbf{a}_x^T \mathbf{S}_i \mod q$.
- Output the sum $\lfloor c_{x,1} \rfloor_{P_1 P_2} + P_2 c_{x,2} + P_1 P_2^2 c_{x,3} + ... + P_1^{\ell-2} P_2^{\ell-1} c_{x,\ell} \mod P$.

Intuition of the construction. What is going on here? Changes from the secure instance are highlighted. Each of the "LWE" instances works mod $q = P_1 P_2 P_1$ and overlaps on the top and bottom (if applicable) by P_1. So the top of each fresh instance hides the "noise bits" of the instance above it. In addition, the "overlap" bits of the $c_{x,1}$ terms are hidden by the $c_{x,2}$ term. So there is some kind of circular security-like thing going on here. Unfortunately, we do not know how to prove this from standard LWE, but we can make a relatively simple and straightforward assumption from which we can prove security (in conjunction with standard LWE). As before, we could choose $\mathbf{S}$ to be from a larger space (i.e. uniform over $\mathbb{Z}_q$), and our implementation even does this, but it would be more difficult to reason about security since there would be more "overlaps".

The main advantage of this construction is it essentially lets us halve the number of rounds needed from our secure construction when we compute the PRF using MPC. For every $c_{x,i}$ term we either need to round or compute a $\mod q$ operation–but not both. Previously, we needed to do both for the $c_{x,1}$ term, which blew up the number of MPC rounds required.

A security proof with a new assumption. We can actually prove this construction secure if we are willing to use a non-standard assumption. We call this new problem the *Overlapping LWR problem.*

Definition 4.4. Overlapping LWR. Consider parameters P_1, P_2, and q, where we set $P_1 = 2^{p_1}$, $P_2 = 2^{p_2}$, and $q = P_1 P_2 P_1$. We also have a security parameter λ, a lattice dimension n based on λ, and the requirement that P_1 and P_2 are superpolynomially large in λ. Let Ψ be a distribution over $\mathbb{Z}^n$. The overlapping learning with rounding problem $(q, n, m, P_1, P_2, \Psi)$-OLWR is to distinguish between the following pairs of distributions:

$$\{\mathbf{A}, \lfloor \mathbf{A}\mathbf{s}(\mathbb{Z}) \rfloor_p + P_2 (\mathbf{A}\mathbf{s} \mod q) \mod P_1^2 P_2^2\}$$
$$\text{and} \qquad \{\mathbf{A}, \mathbf{u}\},$$

where $m = \text{poly}(n)$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{s}_1, \mathbf{s}_2 \leftarrow \Psi$, and $\mathbf{u} \leftarrow \mathbb{Z}_{P_1^2 P_2^2}^m$. Note that we are explicitly *not* computing mod q within the rounding operation.

It turns out, if we assume the overlapping LWR problem to be true, we can prove that our above construction is secure. We state this with the theorem below:

Theorem 4.5. *Let P, n, q, p_1, P_1, p_2, P_2, ℓ, n, and E be parameters as defined above. Let Ψ denote the distribution over $\mathbb{Z}^n$ defined by sampling each entry as an integer uniformly at random in the range $[0, P_1]$ and ψ be the distribution over Z defined by sampling an integer uniformly at random in the range $[0, P_1]$. Any adversary that can win the PRF security game for F_{ASEC} as defined in Definition 2.2 with non-negligible advantage ε can be used to solve the $(q, n, m, P_1, P_2, \Psi)$-OLWR problem or the $(\mathbb{Z}_q, n, \psi, \Psi)$-LWE problem with advantage polynomial in ε assuming that H is a random oracle.*

We sketch this proof in Appendix B. We believe that this overlapping LWR assumption is quite interesting and merits further study.

A construction far from a security proof. We next outline a construction where we exploit overlapping over all of the terms, not just the first two. As before, our construction takes as setup parameters a security parameter λ and some prime P, and then includes other integer parameters n, p_1, p_2, ℓ, n, and E. We have the same parameter requirements (and needed hash function) as our previous construction. The secret key of our PRF F_{FSEC} is a matrix $\mathbf{S} \in \mathbb{Z}_q^{n \times \ell}$ sampled so that each entry is uniform in $[0, ..., P_1/n]$. We denote $\mathbf{S}_i$ to be the ith column of $\mathbf{S}$, and we again note that we could sample $\mathbf{S}$ from a larger space, as before.

We formally define our PRF $F_{\text{FSEC}} : \{0,1\}^k \times \mathbb{Z}_q^{n \times \ell} \to \mathbb{Z}_p$ as follows:

- Set $\mathbf{a}_x \in \mathbb{Z}_q^n = H(x)$.
- Set $c_{x,1} = \mathbf{a}_x^T \mathbf{S}_1$ over $\mathbb{Z}$.
- For $1 < i < \ell$, let $c_{x,i} \in \mathbb{Z} = \mathbf{a}_x^T \mathbf{S}_i$ over $\mathbb{Z}$.
- Set $c_{x,\ell} \in \mathbb{Z}_q = \mathbf{a}_x^T \mathbf{S}_i \mod q$.
- Output the sum $\lfloor c_{x,1} \rceil_{P_1 P_2} + P_2 c_{x,2} + P_1 P_2^2 c_{x,3} + ... + P_1^{\ell-2} P_2^{\ell-1} c_{x,\ell} \mod P$.

Construction explanation. Once again, we highlighted changes from the previous construction. The basic idea here is that we try to take our circular security idea to the extreme. This construction is quite efficient when computed in MPC since we only need to do one mod q operation and one rounding operation.

At first glance, this construction may seem similar to "vanilla" LWR with a very high modulus to "noise" ratio, which would obviously not be good. This is because we can write the construction's outputs as something very close to $\lfloor \mathbf{a}_i (\mathbf{S}_1 + P_2 \mathbf{S}_2 + ...) \rceil_{P_1^{\ell-1} P_2^{\ell}} \mod P$. However, the fact that we are computing a mod q operation in the middle throws chaos into this representation and makes it seemingly unlikely that standard attacks on overstretched LWE will not work here. Attacks on this aggressive construction may have more in common with approaches that might be used to attack the "dark matter" PRF in [11] than LWE itself.

Switching the role of the samples and key. In our constructions in this section, we have used samples (i.e. the outputs of the hash function) that are vectors and keys that are matrices. This was done solely because we could prove security; we know of no attacks if we use samples that are matrices and keys that are vectors (i.e. switching from $\mathbf{a}_i$ and $\mathbf{S}$ to $\mathbf{A}_i$ and $\mathbf{s}$). It is also possible that this could bring about efficiency improvements since we would need to deal with fewer secret bits.

5 Implementation and evaluation

In order to better understand the practical performance of our lattice-based PRF constructions when evaluated in MPC, we implemented and evaluated them using the MP-SPDZ framework [39], which is a framework for evaluating MPC protocols that comes with its own high-level programming language and supports a variety of base MPC protocols. The source code can be found at https://github.com/torusresearch/MP-SPDZ/blob/lattice-prf/Programs/Source/ in files `lattice_prf*.mpc`.

MPC setup. We configure MP-SPDZ to use `mal-shamir` as the base MPC protocol, which is a maliciously secure MPC protocol based on Shamir secret sharing that requires an honest majority, and thereby satisfies our robustness requirement. Moreover, it supports prime field message spaces, which suits the message space of our PRF construction. We run our experiments between 3 parties and with security against 1 corrupted party, and we use a prime field $\mathbb{Z}_{p'}$ as the native message space, where p' is a 256-bit prime.

PRF parameters. We choose the lattice PRF parameters q and p as powers of 2, with varying concrete values during the experiments (e.g., $q = 2^{12}$ and $p = 2^8$). Furthermore, we set $l = \lceil (\log_2(p') + \Delta_s)/\log_2(p) \rceil$, where $\Delta_s = 40$ controls the statistical distance between F_{COMP} and F_{RO}.

Key generation. For generating the master PRF key, the parties need to sample secret shares of a uniformly random $\mathbf{k} \in \mathbb{Z}_q^m$. As there is no method for directly sampling secret-shared values modulo q over the message space $\mathbb{Z}_{p'}$, we rely on random bit sampling [47]. Concretely, for each entry of $\mathbf{k}$, we sample $\log_2 q$ shared random bits and then accumulate them to obtain a shared random integer in $\mathbb{Z}_q$.

Computation modulo q. Evaluating the PRF in MPC requires computing the matrix-vector product $H(x) \cdot \mathbf{k}$ modulo q, where $H(x)$ is a public matrix and $\mathbf{k}$ is a secret-shared vector.

We first compute the matrix-vector product $H(x) \cdot \mathbf{k}$ over the prime field $\mathbb{Z}_{p'}$. Note that p' must be large enough so that the computation does not wrap. Per row of $H(x)$, this involves m cleartext-ciphertext multiplications and $m-1$ ciphertext-ciphertext additions, which can all be done locally. Let $\mathbf{y} \in \mathbb{Z}_{p'}^l$ denote the resulting vector.

Next, we compute each entry of $\mathbf{y}$ modulo q. Since q is a power of 2, this can be done using algorithm `sint.mod2m` of MP-SPDZ, which runs in $1 + \lceil \log_2^2(q) \rceil$ online rounds and makes use of pre-shared random bits.

Rounding operation. Given the vector $\mathbf{y}' = H(x) \cdot \mathbf{k} \bmod q$, we next compute the rounding operation $\lfloor \mathbf{y}' \rceil_p$. Concretely, this means we need to map each element y'_i of $\mathbf{y}'$ to the largest integer j such that $j * q/p \leq y'_i$. Since q and p are powers of 2, this is equivalent to cutting off the $\log_2(q/p)$ lowest order bits of y'_i. In MP-SPDZ this can be done using algorithm `sint.right_shift`, which runs in $1 + \lceil \log_2^2(q/p) \rceil$ online rounds and makes use of pre-shared random bits.

Composition. Finally, we compose the entries of the rounded output vector $\lfloor \mathbf{y}' \rceil_p$ into a single uniform random value $y'' \in \mathbb{Z}_{p'}$ by evaluating M_{COMP}. This involves plaintext-ciphertext multiplications and ciphertext-ciphertext additions, which can all be done locally.

Total round count. The total round count for computing $F_{\text{COMP}}(x)$ in MPC is $2 + \log_2^2(q) + \log_2^2(q/p)$. For $q = 2^{12}$ and $p = 2^8$, this results in a MPC PRF protocol running in 8 online rounds, assuming sufficiently many shared random bits have been generated during preprocessing.

5.1 Parameter selection

In the following, we describe the lattice parameters that we use for our evaluation. As in all practical lattice-based cryptosystems, parameters are derived from what we know about best possible attacks rather than theoretical reductions, so we do not carry over parameters from our theorems here. Instead, our parameter selection is inspired by the parameters of Frodo [13] with 128 bit classical security, as this scheme relies on a similar assumption. We emphasize that Frodo was meant to be a very conservative construction as well.

As the lattice dimension we choose a fixed value of $m = 512$. We note that the lattice dimension does not affect the complexity of the multi-party computation. This is because only the size of the secret key $\mathbf{k}$ and the hash output $H(x)$ depend on m. However, as the vector product $H(x) \cdot \mathbf{k}$ collapses these vector elements onto a single element, and this is computed locally at the start of the protocol, the interactive part of the protocol does not depend on m at all.

For the modulo parameter q and the rounding parameter p, we use the two parameter sets $(q,p) = (2^{12}, 2^8)$ and $(q,p) = (2^{32}, 2^{24})$. Here, smaller values optimize for round complexity, while larger values optimize for preprocessing complexity due to the lower number of pre-shared random bits required. We also distinguish between whether we use the regular PRF construction F_{COMP} (cf. Section 3.3), the optimized secure PRF construction F_{SEC} (cf. Section 4.3), or the optimized conjectured constructions F_{ASEC} and F_{FSEC} (cf. Section 4.4). We list all considered protocol instances in Table 1.

Instance	Construction	Param q	Param p
REG_{12}	F_{COMP}	2^{12}	2^8
REG_{32}	F_{COMP}	2^{32}	2^{24}
OPT_{12}	F_{SEC}	2^{12}	2^8
OPT_{32}	F_{SEC}	2^{32}	2^{24}
$\mathsf{CON}_{12}^{\text{ASEC}}$	F_{ASEC}	2^{12}	2^8
$\mathsf{CON}_{32}^{\text{ASEC}}$	F_{ASEC}	2^{32}	2^{24}
$\mathsf{CON}_{12}^{\text{FSEC}}$	F_{FSEC}	2^{12}	2^8
$\mathsf{CON}_{32}^{\text{FSEC}}$	F_{FSEC}	2^{32}	2^{24}

Table 1: Parameterized instances of our PRF constructions used for evaluation.

5.2 Evaluation

We measure the performance of our distributed PRF protocol with respect to the different instances defined in Section 5.1 and also in comparison with other existing distributed PRF protocols. We run our experiments on a single machine with an M1 Pro CPU, 32GB RAM, and with local communication (e.g., without network latency) and mostly focus on measuring performance of the online phase without preprocessing if not stated otherwise.

Comparison between instances. We first measure and compare the performance of the different protocol instances defined in Section 5.1. The results are shown in Table 2. We observe that larger values for q and p lead to less preprocessing at the cost of a higher online round count. Furthermore, the optimized variant further reduces the preprocessing and communication cost, at the expense of more computation due to the larger secret key.

Comparison with other protocols. We compare our provably secure distributed PRF protocols with existing constructions in Table 3, relying on the implementations from [34]. Most of the other protocols are designed for producing multiple field elements per protocol evaluation. Notably, GMiMC does not support output size 1 and for Rescue reducing the output size to 1 increases the round complexity. Hence, we chose an output size of 2 for these schemes.

Overall, our protocol compares favorably in terms of running time and round complexity. However, we note that due to the large number of random bits consumed by our protocol, it requires more communication, especially during the preprocessing phase. We note that the amount of preprocessing communication is still comfortably within the realms of feasibility and does not limit the applicability of our protocol. It is an interesting open question how to produce a large number of pre-shared random bits with less communication. For many applications of DKD systems, like cryptocurrency transac-

Protocol	Time (ms)	Data (MB) Online / Total	Rounds	Bits
Provable security				
REG_{12}	11.34	0.41 / 6.01	8	4625
REG_{32}	8.06	0.43 / 3.85	10	2405
OPT_{12}	24.79	0.33 / 3.85	8	2753
OPT_{32}	12.86	0.36 / 2.79	10	1541
Conjectured security				
$\mathsf{CON}_{12}^{\mathrm{ASEC}}$	24.49	0.32 / 3.73	5	2680
$\mathsf{CON}_{32}^{\mathrm{ASEC}}$	12.20	0.34 / 2.56	6	1428
$\mathsf{CON}_{12}^{\mathrm{FSEC}}$	22.89	0.01 / 0.31	5	125
$\mathsf{CON}_{32}^{\mathrm{FSEC}}$	11.47	0.03 / 0.33	6	185

Table 2: Measurements for running one evaluation of each of our PRF protocol instances. *Time* is the time required to run the protocol on a single machine without network latency. *Data* is the global amount of data sent between all parties in the online phase and including preprocessing, respectively. *Rounds* is the number of online communication rounds. *Bits* is the number of pre-shared random bits consumed.

tions or other digital proof claims, requests are triggered by a user interacting with a wallet. This typically means that having low latency is extremely important, but requests are relatively infrequent and there is ample time for preprocessing.

6 Application to Distributed Key Management

In the following we propose a model for distributed key management in the Universal Composability Framework [15]. We first introduce an abstract MPC functionality $\mathcal{F}_{\mathsf{MPC}}$ that our protocols will depend on. Then, we model the ideal functionality of DKM with a key distribution $\mathcal{K}$, define a real protocol based on a PRF F, and show that the real protocol realizes the ideal functionality in the $\mathcal{F}_{\mathsf{MPC}}$-hybrid model if the key distribution $\mathcal{K}$ is indistinguishable from the output distribution of the PRF F. Finally, as an example instantiation, we show that the proposed DKM protocol instantiated with one of our distributed PRF protocols efficiently realizes distributed key management for ECDSA signing.

We work in the universal composability framework [15] and assume authenticated private channels between parties. Furthermore, we assume access to the following ideal MPC functionality, $\mathcal{F}_{\mathsf{MPC}}$, which abstractly represents an MPC protocol library such as MP-SPDZ [39]. We require that any realization of $\mathcal{F}_{\mathsf{MPC}}(P,t)$ is secure against static corruption of at most t parties of P.

Functionality 6.1 ($\mathcal{F}_{\mathsf{MPC}}(P,t)$, Multi-party computation)**.** This functionality is parametrized by the set of servers P and the corruption threshold t. The simulator $\mathcal{S}$ may statically corrupt up to t parties in P. We denote by $[x]$ the identifier of a stored secret value x.

Evaluate: On message $(\texttt{eval}, \mathsf{sid}, f, [k], x)$ from all parties in P, compute $y \leftarrow f(k,x)$, store y, and send $(\texttt{result}, \mathsf{sid}, [y])$ to all parties in P.

Open: On message $(\texttt{open}, \mathsf{sid}, [x], u)$ from all parties in P, send $(\texttt{value}, \mathsf{sid}, x)$ to party u.

6.1 Ideal functionality

In the following, we first define an ideal functionality for Distributed Key Management involving a set of servers P and a set of users U, where the servers create and manage secret keys on behalf of the users. The users can ask the servers to evaluate a fixed set of algorithms, $\mathcal{F}_{\mathsf{Eval}}$, on their secret keys, where $\mathcal{F}_{\mathsf{Eval}}$ depends on the application and may, for example, allow the users to create digital signatures using the managed secret keys as input.

Functionality 6.2 ($\mathcal{F}_{\mathsf{DKM}}(P,t,U,\mathcal{K},\mathcal{F}_{\mathsf{Eval}})$, Distributed Key Management)**.** This functionality is parametrized by the set of servers P, the corruption threshold t, the set of users U, the key distribution $\mathcal{K}$, and the set of supported key-dependent algorithms $\mathcal{F}_{\mathsf{Eval}}$. The simulator $\mathcal{S}$ may statically corrupt up to t parties in P and any subset of parties of U, which it announces at the start of the protocol.

Setup: On receiving $(\texttt{init})$ from all parties in P, store $(\texttt{ready})$, and send $(\texttt{ready})$ to all parties in P.

Eval: On receiving $(\texttt{eval}, \mathsf{sid}, f, x)$ from party $u \in U$, assert that $(\texttt{ready})$ is stored and $f \in \mathcal{F}_{\mathsf{Eval}}$. If this is the case, do the following.

1. Check if there is a storage entry $(\texttt{key}, u, k)$, for some key k. If there is no such entry, sample $k \leftarrow \mathcal{K}$, and store $(\texttt{key}, u, k)$.
2. Call $k \leftarrow \mathsf{Key}(u)$, compute $y \leftarrow f(k,x)$ and all parties in U, and send $(\texttt{result}, \mathsf{sid}, y)$ to u.

6.2 Real protocol

The real protocol is defined in the $\mathcal{F}_{\mathsf{MPC}}$-hybrid model and defines a simple interaction between the users U and the servers P where in each protocol instance a user instructs the servers to run an algorithm $f \in \mathcal{F}_{\mathsf{Eval}}$ on its derived key, and the servers use $\mathcal{F}_{\mathsf{MPC}}$ to perform this request in MPC.

Protocol 6.3 ($\pi_{\mathsf{DKM}}(P,t,U,F,\mathcal{F}_{\mathsf{Eval}})$)**.** This protocol is parametrized by the set of servers P, the corruption thresold t, the set of users U, the PRF F, and the set of supported key-dependent algorithms $\mathcal{F}_{\mathsf{Eval}}$. It makes use of functionality $\mathcal{F}_{\mathsf{MPC}}(P,t)$.

Setup:

Protocol	**Computation** (ms)	**Communication** (MB) Online / Total	**Rounds**	**Time with network latency** (in seconds, extrapolated)
Ciminion [24]	31.93	0.11 / 0.36	283	28.33
GMiMC [3]	59.04	0.26 / 0.84	670	67.06
HadesMiMC [33]	47.81	0.18 / 0.60	466	46.65
Hydra [34]	21.42	0.07 / 0.24	140	14.02
MiMC [1]	33.18	0.13 / 0.46	331	33.13
Rescue [5]	26.35	0.28 / 0.97	124	12.43
Our protocol REG_{12}	11.34	0.41 / 6.01	8	0.81
Our protocol REG_{32}	8.06	0.43 / 3.85	10	1.01
Our protocol OPT_{12}	24.79	0.33 / 3.85	8	0.82
Our protocol OPT_{32}	12.86	0.36 / 2.79	10	1.01

Table 3: Comparison of existing distributed PRF protocols with our provably-secure protocols. *Time with network latency* estimates the execution time with an assumed round trip time of 100 milliseconds.

1. On message $(\texttt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i \in P$ checks whether there is an entry $(\texttt{prfkey}, \cdot)$ in memory. If not, $\mathcal{P}_i$ sends $(\texttt{eval}, \mathsf{sid}, F.\mathsf{KeyGen})$ to functionality $\mathcal{F}_{\mathsf{MPC}}$.
2. On message $(\texttt{result}, \mathsf{sid}, \mathbf{k})$ from $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{P}_i$ stores $(\texttt{prfkey}, \mathbf{k})$.

Eval:

1. On message $(\texttt{eval}, \mathsf{sid}, f, x)$ from the environment $\mathcal{Z}$, a user $u \in U$ sends $(\texttt{eval}, \mathsf{sid}, f, x)$ to each party in $\mathcal{P}_i \in P$.
2. On message $(\texttt{eval}, \mathsf{sid}, f, x)$ from a user $u \in U$, each party $\mathcal{P}_i \in P$ asserts that there is an entry $(\texttt{prfkey}, \mathbf{k})$ in memory, for some $\mathbf{k}$, and that $f \in \mathcal{F}_{\mathsf{Eval}}$. It then sends $(\texttt{eval}, \mathsf{sid}, g, \mathbf{k}, u, x)$ to $\mathcal{F}_{\mathsf{MPC}}$, where $g(\mathbf{k}, u, x) = f(F.\mathsf{Eval}(\mathbf{k}, u), x)$.
3. On message $(\texttt{result}, \mathsf{sid}, y)$ from $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{P}_i$ sends $(\texttt{open}, \mathsf{sid}, y, u)$ to $\mathcal{F}_{\mathsf{MPC}}$.
4. On message $(\texttt{value}, \mathsf{sid}, y)$ from $\mathcal{F}_{\mathsf{MPC}}$, u outputs $(\texttt{result}, y)$.

6.3 Security proof

The following thereom establishes that protocol π_{DKM} with key distribution $\mathcal{K}$ realizes functionality $\mathcal{F}_{\mathsf{DKM}}$ with PRF F, for any fixed set of algorithms $\mathcal{F}_{\mathsf{Eval}}$, if the key distribution $\mathcal{K}$ is indistinguishable from the output distribution of F.

Theorem 6.4. *Let P and U be sets of parties, and t be a corruption threshold. Let F be a PRF and $\mathcal{K}$ be a key distribution such that the output distribution of F is indistiguishable from $\mathcal{K}$. Let $\mathcal{F}_{\mathsf{Eval}}$ be a set of polynomial-time algorithms. Then, protocol $\pi_{DKM}(P,t,U,F,\mathcal{F}_{\mathsf{Eval}})$ UC-realizes functionality $\mathcal{F}_{\mathsf{DKM}}(P,t,U,\mathcal{K},\mathcal{F}_{\mathsf{Eval}})$ in the $\mathcal{F}_{MPC}(P,t)$-hybrid model against a malicious adversary that statically corrupts up to t parties of P and any subset of parties of U.*

The proof follows immediately from the definition of the protocol, the security of the MPC functionality in the specified corruption setting, and the indistiguishability of the key distribution $\mathcal{K}$ and the output distribution of the PRF F. A more explicit version of the proof can be found in appendix A.2.

6.4 Example application: Scalable distributed key management for ECDSA signing using our distributed PRF protocol

In the following, we show, as an example, that our distributed PRF protocol in combination with a realization of $\mathcal{F}_{\mathsf{MPC}}$ can be used to efficiently realize a distributed key management system that supports ECDSA signing in MPC.

Ideal functionality $\mathcal{F}_{\mathsf{DKM}}^{\mathsf{ECDSA}}$. We first define the ideal functionality of a distributed key management system for ECDSA. Let ECDSA denote the ECDSA signature scheme over group $\mathcal{G}$ with generator G and prime order p. Define the following algorithms.

PubKey**:** On input (k,x), parse k as (sk,pk), and return pk.

$\mathsf{PrivKey}$**:** On input (k,x), parse k as (sk,pk), and return sk.

Sign**:** On input (k,x), parse k as (sk,pk), compute $\sigma \leftarrow \mathsf{ECDSA.Sign}(sk,x)$, and return σ.

Let $\mathcal{F}_{\mathsf{ECDSA}} = \{\mathsf{PubKey}, \mathsf{PrivKey}, \mathsf{Sign}\}$. The ideal functionality of a distributed key management system for ECDSA is defined as

$$\mathcal{F}_{\mathsf{DKM}}^{\mathsf{ECDSA}}(P,t,U) = \mathcal{F}_{\mathsf{DKM}}(P,t,U,\mathsf{ECDSA.KeyGen},\mathcal{F}_{\mathsf{ECDSA}}) \ .$$

Real protocol $\pi_{\mathsf{DKM}}^{\mathsf{ECDSA}}$. Let LPRF denote our lattice-based PRF instantiated over $\mathbb{Z}_p$. Define Derive as an algorithm that

takes as input a PRF key k and a user identifier u, and outputs an ECDSA keypair (sk, pk), as follows. First, compute $sk = \mathsf{LPRF.Eval}(k,u)$. Then compute $pk = sk \cdot G$. Output (sk, pk). Define $F_{\mathsf{ECDSA}} = (\mathsf{LPRF.KeyGen}, \mathsf{Derive})$. Now, the real protocol for ECDSA DKM is defined as

$$\pi_{\mathsf{DKM}}^{\mathsf{ECDSA}}(P,t,U) = \pi_{\mathsf{DKM}}(P,t,U,F_{\mathsf{ECDSA}},\mathcal{F}_{\mathsf{ECDSA}}) \ .$$

Security. By Thereom 3.2, we know that the output distribution of LPRF is indistiguishable from a uniform random distribution over $\mathbb{Z}_p$. This, however, is exactly the secret key distribution of ECDSA over $\mathcal{G}$. It follows that the output distribution of F_{ECDSA} is indistiguishable from the distribution of ECDSA.KeyGen and, hence, by Thereom 6.4 we have that $\pi_{\mathsf{DKM}}^{\mathsf{ECDSA}}(P,t,U)$ realizes $\mathcal{F}_{\mathsf{DKM}}^{\mathsf{ECDSA}}(P,t,U)$.

Realizing $\mathcal{F}_{\mathsf{MPC}}$. To fully realize $\mathcal{F}_{\mathsf{DKM}}^{\mathsf{ECDSA}}$ one will also need to instantiate $\mathcal{F}_{\mathsf{MPC}}$ with a real world MPC framework such as MP-SPDZ [39]. Furthermore, existing optimized MPC-based ECDSA signing protocols [16, 21, 25, 26, 36] should be considered for efficient realization of ECDSA signing in MPC.

References

[1] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 191–219, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[2] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and fhe. Cryptology ePrint Archive, Paper 2023/232, 2023. https://eprint.iacr.org/2023/232.

[3] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for mpc, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *Computer Security – ESORICS 2019*, pages 151–171, Cham, 2019. Springer International Publishing.

[4] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 430–454, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[5] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Transactions on Symmetric Cryptology*, 2020(3):1–45, Sep. 2020.

[6] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[7] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[8] Mihir Bellare and Phil Rogaway. Random oracle are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[9] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symmetric Cryptol.*, 2017(4):1–38, 2017.

[10] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the legendre PRF and generalizations. *IACR Cryptol. ePrint Arch.*, page 1357, 2019.

[11] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter:. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography*, pages 699–729, Cham, 2018. Springer International Publishing.

[12] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[13] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring!

practical, quantum-secure key exchange from lwe. Cryptology ePrint Archive, Paper 2016/659, 2016. https://eprint.iacr.org/2016/659.

[14] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC'13*, pages 575–584, 2013.

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Paper 2000/067, 2000. https://eprint.iacr.org/2000/067.

[16] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. Cryptology ePrint Archive, Paper 2021/060, 2021. https://eprint.iacr.org/2021/060.

[17] Coinbase WaaS. Coinbase web page. https://www.coinbase.com/cloud/products/waas, November 2023.

[18] Cointelegraph. Crypto custody market reached $448 billion in 2022: Report - Cointelegraph. https://cointelegraph.com/news/crypto-report-the-crypto-custody-market-reached-448-billion-in-2022, February 2024.

[19] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *Theory of Cryptography*, pages 342–362, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[20] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[21] Anders Dalskov, Marcel Keller, Claudio Orlandi, Kris Shrishak, and Haya Shulman. Securing dnssec keys via threshold ecdsa from generic mpc. Cryptology ePrint Archive, Paper 2019/889, 2019. https://eprint.iacr.org/2019/889.

[22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

[23] Dfns. Dfns web page. https://www.dfns.co, July 2023.

[24] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric encryption based on toffoli-gates over large finite fields. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2021.

[25] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ecdsa in three rounds. Cryptology ePrint Archive, Paper 2023/765, 2023. https://eprint.iacr.org/2023/765.

[26] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066, 2019.

[27] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[28] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001.

[29] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[30] Fireblocks. Fireblocks web page. https://www.fireblocks.com, November 2023.

[31] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 295–310. Springer, 1999.

[32] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 34(4):792–807, 1986.

[33] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The hades design strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 674–704, Cham, 2020. Springer International Publishing.

[34] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. From farfalle to megafono via ciminion: The PRF hydra for MPC applications. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of*

Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV, volume 14007 of *Lecture Notes in Computer Science*, pages 255–286. Springer, 2023.

[35] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. Mpc-friendly symmetric key primitives. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 430–443, New York, NY, USA, 2016. Association for Computing Machinery.

[36] Iftach Haitner, Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Paper 2018/987, 2018. https://eprint.iacr.org/2018/987.

[37] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology — CRYPT0' 95*, pages 339–352, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[38] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

[39] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[40] Lit Protocol. Lit protocol web page. https://litprotocol.com, July 2023.

[41] Easwar Vivek Mangipudi and Aniket Pundlik Kate. D-kode: Distributed mechanism to manage a billion discrete-log keys. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, page 308–325, New York, NY, USA, 2023. Association for Computing Machinery.

[42] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. *IACR Cryptology ePrint Archive*, 2013.

[43] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 458–467. IEEE Computer Society, 1997.

[44] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *41st Annual ACM Symposium on Theory of Computing — STOC '09*, pages 333–342. ACM, 2009.

[45] Qredo. Qredo web page. https://www.qredo.com, November 2023.

[46] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05*, pages 84–93. ACM, 2005.

[47] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology – INDOCRYPT 2019*, pages 227–249, Cham, 2019. Springer International Publishing.

[48] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[49] Silence Laboratories. Silence Laboratories web page. https://www.silencelaboratories.com, November 2023.

[50] Web3Auth. Web3Auth web page. https://web3auth.io, July 2023.

[51] Zengo. Zengo web page. https://zengo.com, November 2023.

A Proof of Theorems 2.1 and 6.4

A.1 Proof of Theorem 2.1

Proof. Let $m, n \in \mathbb{N}$, $X = U(\mathbb{Z}_m)$, and $Y = U(\mathbb{Z}_n) \bmod m$. By the definition of the statistical distance, we have

$$\begin{aligned}\Delta(X,Y) &= \frac{1}{2}\sum_{a\in\mathbb{Z}_m} |\Pr[X=a] - \Pr[Y=a]| \\ &= \frac{1}{2}\sum_{a\in\mathbb{Z}_m} |1/m - \Pr[Y=a]| \ .\end{aligned} \tag{1}$$

For $a \in \mathbb{Z}_m$, define $D_a = \{x \in \mathbb{Z}_n : x \bmod m = a\}$. We have

$$\Pr[Y=a] = \sum_{b\in D_a} \Pr[U(\mathbb{Z}_n) = b] = |D_a|/n \ , \tag{2}$$

where

$$|D_a| = \begin{cases} \lceil n/m \rceil & \text{if } a < n \bmod m, \\ \lfloor n/m \rfloor & \text{else.} \end{cases} \tag{3}$$

Next, we determine an upper bound on $|1/m - \Pr[Y=a]|$ by writing out the equation using (2) and (3). For $a < n \bmod m$, we have

$$\begin{aligned}\left|\frac{1}{m} - \Pr[Y=a]\right| &= \left|\frac{1}{m} - \frac{\lceil n/m \rceil}{n}\right| \\ &= \left|\frac{1}{m} - \frac{n+m-n \bmod m}{mn}\right| \le \frac{1}{n} \ ,\end{aligned} \tag{4}$$

and for $a \geq n \bmod m$,

$$\left|\frac{1}{m} - \Pr[Y = a]\right| = \left|\frac{1}{m} - \frac{\lfloor n/m \rfloor}{n}\right| = \left|\frac{1}{m} - \frac{n - n \bmod m}{mn}\right| \leq \frac{1}{n} \ . \tag{5}$$

Finally, by combining (1), (4), and (5), we obtain an upper bound on the statistical distance of X and Y,

$$\Delta(X,Y) \leq \frac{1}{2} \sum_{a \in \mathbb{Z}_m} 1/n \leq m/n \ .$$

□

A.2 Proof of Thereom 6.4

Proof. We have to show that for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that for every environment $\mathcal{Z}$, the environment $\mathcal{Z}$ cannot distinguish a real execution of protocol $\pi_{\mathsf{DKM}}(P,t,U,F,\mathcal{F}_{\mathsf{Eval}})$ with adversary $\mathcal{A}$ from an ideal execution of functionality $\mathcal{F}_{\mathsf{DKM}}(P,t,U,\mathcal{K},\mathcal{F}_{\mathsf{Eval}})$ with simulator $\mathcal{S}$.

Let $\mathcal{A}$ be such an adversary that controls up to t corrupted parties of P and any subset of parties of U in the real protocol execution. We define $\mathcal{S}$ as a simulator that simulates $\mathcal{A}$ in the ideal execution, as follows.

Setup: Whenever the simulator obtains an input $(\texttt{init}, \mathsf{sid})$ on behalf of a corrupted server $\mathcal{P}$, the simulator forwards this message to $\mathcal{A}$. If $\mathcal{A}$ behaves according to the protocol and sends $(\texttt{eval}, \mathsf{sid}, F.\mathsf{KeyGen})$ to functionality $\mathcal{F}_{\mathsf{MPC}}$, the simulator sends $(\texttt{init})$ to $\mathcal{F}_{\mathsf{DKM}}$ on behalf of $\mathcal{P}$. If $\mathcal{F}_{\mathsf{DKM}}$ responds with $(\texttt{ready})$, $\mathcal{S}$ sends $(\texttt{eval}, \mathsf{sid}, F.\mathsf{KeyGen})$ to functionality $\mathcal{F}_{\mathsf{MPC}}$ on behalf of all parties in P, and forwards the response $(\texttt{result}, \mathsf{sid}, \mathbf{k})$ to the adversary $\mathcal{A}$ and stores $(\texttt{prfkey}, \mathbf{k})$.

Eval: Whenever the simulator obtains an input $(\texttt{eval}, \mathsf{sid}, f, x)$ on behalf of a corrupted user u, the simulator forwards this message to $\mathcal{A}$. The simulator then emulates an execution of the Eval protocol, where it plays the role of the honest parties and the adversary $\mathcal{A}$ plays the role of the corrupted parties. Here, the simulator makes use of functionality $\mathcal{F}_{\mathsf{MPC}}$ and the stored PRF key reference $\mathbf{k}$. Since the adversary controls only up to t parties, the user u is guaranteed to receive $(\texttt{value}, \mathsf{sid}, y)$ from $\mathcal{F}_{\mathsf{MPC}}$ as a result. The simulator then sends $(\texttt{eval}, \mathsf{sid}, f, x)$ to $\mathcal{F}_{\mathsf{DKM}}$ on behalf of party u and waits for response $(\texttt{result}, \mathsf{sid}, y)$.

We observe that, by the definition of $\mathcal{F}_{\mathsf{MPC}}$, executions of the real protocol with $\mathcal{A}$ look almost identical to executions of the ideal functionality with $\mathcal{S}$ from the viewpoint of $\mathcal{E}$. The difference is that in the real execution the key $\mathbf{k}$ comes from the output distribution of F while in the ideal execution $\mathbf{k}$ comes from the key distribution $\mathcal{K}$. However, since we chose F and $\mathcal{K}$ such that their distributions are indistiguishable, the environment cannot distinguish the real execution from the ideal execution. □

B Enhanced constructions

Due to space constraints, this section is only available in the full version of this paper (ia.cr/2023/1254).