Practice-Oriented Instances of Deterministic LPN

Carlos Cid[1,3][0000−0001−5761−8694], Alex Davidson[2][0000−0001−9157−3821], and Atharva Phanse[1(✉)][0009−0006−1914−3484]

[1] Simula UiB, Norway
[2] Faculdade de Ciências at Universidade de Lisboa, Portugal
[3] Okinawa Institute of Science and Technology Graduate University, Japan
carlos@simula.no,aadavidson@ciencias.ulisboa.pt,atharva@simula.no

Abstract. We develop a framework for instantiating and implementing the Learning Parity with Noise (LPN) assumption, via a deterministic noise paradigm. Our approach allows high flexibility in the choice of error rate (τ), mirroring the functionality of Bernoulli sampling in the standard LPN. This improves upon previous deterministic frameworks which are fairly rigid in the choice of τ, or depend on non-cryptographic sources of physical noise. We give a number of theoretical relations between existing deterministic frameworks and our new design. In addition, we build new instantiations of τ-Unbalanced Weak Pseudorandom Functions, based on novel compositions and designs of diffusion and folding functions. Our choices of building blocks take from both symmetric cryptographic primitives and low-depth Boolean formulae. Finally, we implement our framework with a selection of diffusion tools, and show that native hardware instructions in common CPUs give rise to instantiations that are highly efficient. All in all, we provide a firm basis for real-world deployment of recent LPN-based applications — such as those related to VOLE proof systems and MPC — even in scenarios where high-quality randomness sources do not exist, e.g. resource-constrained devices.

Keywords: LPN · Weak Pseudorandom Functions · Deterministic Noise

1 Introduction

The Learning Parity with Noise (LPN) problem [15] is a widely studied topic in complexity theory and has a rich history in cryptography. Given a uniform distribution $\mathcal{U}$ over $\mathbb{Z}_2$, and a Bernoulli distribution Ber_τ with parameter $\tau < 1/2$, the decisional variant of LPN asks one to distinguish samples of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e) \in \mathbb{Z}_2^\ell \times \mathbb{Z}_2$ given that $\boldsymbol{a}, \boldsymbol{s} \hookleftarrow \mathcal{U}^\ell$, and $e \leftarrow \mathsf{Ber}_\tau$, from $(\boldsymbol{a}, u) \hookleftarrow (\mathcal{U}^\ell, \mathcal{U})$. The problem is known to be NP-hard in the worst case [46], while Blum, Kalai, and Wasserman [16] showed that random instances could be solved with subexponential $O(2^{\ell/\log \ell})$ data and time complexity.[4] Importantly, meaningful

[4] Recently, so-called *linear test* frameworks have arisen aiming to capture this attack and subsequent cryptanalytic advances in a unified adversarial strategy [4,27,22].

quantum speed-ups for LPN are not known to exist, which makes it a potential building block for post-quantum cryptography.

Since 2001, LPN has been used to instantiate secure identification protocols [50,52], public-key encryption [39], PRNGs [15], and more [60]. More recently, several new applications of LPN have been proposed, including vector oblivious linear evaluation (VOLE) protocols [4], multi-party computation protocols [13], post-quantum signatures [9] and multi-party homomorphic secret sharing [28]. From a theoretical perspective, certain low-depth pseudorandom functions (PRF) are intrinsically linked to instances of the LPN assumption [2,19,23], and by extension, enjoy various practical use-cases. Alongside this *renaissance*, a variety of related problems have emerged that aid the building of new constructions. These include a ring-based variant (Ring-LPN) [48], as well as variants of the problem instance, such as variable-density [22] and sparse LPN [32], that are still conjectured to remain hard.

Motivation for studying determinism in LPN. Beyond the applications mentioned, the simplicity of the LPN assumption can make it very attractive for lightweight applications, particularly for authentication in resource-constrained environments. This is exemplified by the *HB family* of protocols [50], with follow-up variations including [52,38,44,58] (see Appendix A for an overview). However, prior work [5] has noted that a main drawback of LPN-based protocols is that Bernoulli sampling requires significant computational resources, which can be, in general, monetarily expensive. In addition, when considering authentication protocols, the cost of the Bernoulli sampling is often on the prover (*tag*) device, which is typically weaker. Despite such drawbacks, the strong security guarantees of the LPN assumption have motivated researchers to investigate ways to overcome its practical limitations. To address the limitations associated with the Bernoulli sampling in real-world applications, the following question arises:

Can we devise alternative interpretations of LPN that do not require native sources of entropy and randomness for producing secure instances?

Derandomization is a standard approach in cryptography to mitigate reliance on randomness and address these limitations. In the case of LWE, the Learning with Rounding (LWR) problem [6] provides a counterpart that allows producing LWE-like samples, but effectively with deterministic errors. Various works have shown the equivalence of LWR to LWE, for a variety of parameter selections [20,3], meaning that many LWE-based cryptographic applications may be redesigned in the LWR paradigm, and deployed where access to random sampling from LWE error distributions is not possible.

Inflexibility in deterministic noise generation. In the case of LPN, various works have shown this to be also possible. Recent literature on weak PRFs has already made connections between constructions and deterministic instances of LPN (which we will refer to as *DLPN*) with specific noise rates. For example, the weak PRF candidate defined as:

$$\mathsf{PRF}(\boldsymbol{k}, \boldsymbol{x}) = (\langle \boldsymbol{k}, \boldsymbol{x} \rangle \pmod 2 + \langle \boldsymbol{k}, \boldsymbol{x} \rangle \pmod 3) \pmod 2,$$

proposed in [19] is equivalent to a deterministic LPN instance with noise rate $\tau = 1/3$. However, modifying such candidates to cater for different noise rates appears difficult.[5] . A formal framework called LPN with *simple* deterministic noise was proposed [23], where LPN samples are constructed as $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + g(k, \boldsymbol{a})$, for a secret $k \leftarrow\!\!\$\ \{0,1\}^\lambda$. While such a framework would appear to provide more flexibility, the presence of an additional secret k is cumbersome: it requires additional storage on the part of small devices, and running of a secure process for embedding such secrets onto chips. From a more theoretical angle, the function g does not appear to admit simple configuration of the *error parameter* $\tau < 1/2$ (without modifying the effective *security parameter* λ), which means that a new function effectively has to be configured for any given choice of noise rate. We believe this provides a practical barrier to use them for instantiating practical and meaningful LPN applications, where variable noise rates may be required for maintaining functionality and security.

On the other hand, a recent line of literature has shown links between deterministic creation of LPN samples, with a paradigm known as Learning Parity with Physical Noise (LPPN) [55]. In LPPN, the noise is sampled from physical phenomena, such as via inexact implementations of inner-products that can arise during the engineering process of seemingly identical hardware chips. Although the approach offers a promising way to avoid random error sampling in LPN, hardware-based cryptanalysis has revealed effective attacks, suggesting that achieving strong cryptographic security remains challenging [11,53,12].

Practical, deterministic instances for configurable noise. In summary, to migrate a range of LPN-based schemes to real-world applications implemented in environments with constrained entropy sources, we need the following:

- to design and instantiate a deterministic noise framework for LPN, with the ability to configure the noise rate flexibly, ideally for any $0 < \tau < 1/2$;
- to provide concrete realisations of deterministic noise sampling functions $g_\tau(\boldsymbol{a}, \boldsymbol{s})$ that enable producing LPN samples that satisfy cryptographic guarantees of security, while producing noise rates very close to τ; and,
- that such functions must be efficient to execute, even in constrained environments.

In principle, DLPN instances that satisfy these constraints would allow deploying advanced cryptographic primitives across a range of computing scenarios. Note that it is important to provide a distinction between theoretical low-complexity techniques, and those that are very efficient in practice in hardware and software. In particular, while recent developments in low-depth pseudorandom functions appear to provide promising avenues forward for efficient implementations, equally as interesting are PRF constructions that rely heavily on real-world optimisations (such as embedded hardware-level instructions).

Overview and contributions. In this work, we first develop a modified deterministic noise framework (Section 3) for LPN samples of the form:

$$c_i = \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + g_\tau(\boldsymbol{a_i}, \boldsymbol{s}) \pmod 2$$

[5] In the case of [19], modifying the moduli produces noise rates that converge to 1/2.

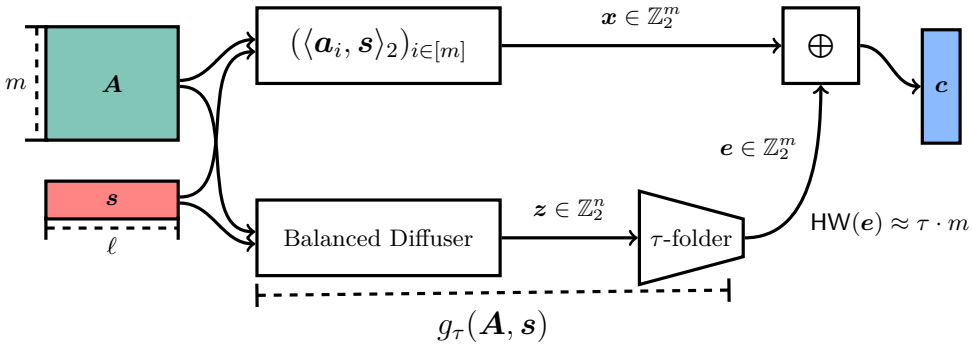

Fig. 1. Diagrammatic representation of DLPN instance generation.

where $g_\tau : \mathbb{Z}_2^{m\times\ell} \times \mathbb{Z}_2^\ell \to \{0,1\}^m$. When $m = 1$, we construct g_τ such that $\Pr\big[1 \leftarrow g_\tau(\boldsymbol{a}, \boldsymbol{s} : \boldsymbol{a}, \boldsymbol{s} \leftarrow\$ \mathbb{Z}_2^\ell)\big] \approx \tau$. For general m, we consider $\boldsymbol{A} \leftarrow\$ \mathbb{Z}_2^{m\times\ell}$, and construct g_τ such that $\mathsf{HW}(g_\tau(\boldsymbol{A}, \boldsymbol{s})) \approx \tau \cdot m$. We do so by composing a *balanced diffuser* function with a τ-dependent *folding* function.

Secondly, we demonstrate a possible construction for τ-folder (F_τ) that accepts balanced binary strings of length n (Section 4), and produces binary strings of length m with hamming weight $\tau \cdot m$. Our formulation of F_τ follows similar ideas to those used in the development of the Tribes [2] and SIPSER [23] functions used at the heart of low-depth pseudorandom functions. However, our approach, which we refer to as *universal* Tribes, allows configuring the function flexibly, dependent on the desired value of the parameter τ.

For instantiating the balanced diffuser function ($\mathsf{Diffuse} : \mathbb{Z}_2^{\ell\times m} \times \mathbb{Z}_2^\ell \to \mathbb{Z}_2^\lambda$) we consider several approaches ranging from full pseudorandom functions, to more streamlined approaches that conjecture security based on the "full diffusion" property of a sufficiently non-linear permutation. In principle, we note that algebraic attacks that expose the insecurity of the original Tribes function stemmed from the fact that Diffuse was essentially implemented as a linear function [18]. We argue that non-linear balanced functions can serve to instantiate this paradigm, and we discuss relevant cryptanalysis accordingly. Bringing the previous two results together, in Lemma 3, we show that implementing $g_\tau = \mathsf{F}_\tau \circ \mathsf{Diffuse}$ directly from a PRF constructs DLPN samples that are indistinguishable from standard LPN samples with error-rate τ.

To highlight performance variability in our approach to building DLPN instances, we provide experimental analysis (Section 6) that compares several ways of implementing Diffuse on two different platforms. We observe that, in general, all Diffuse instantiations that we devise outperform constructions based on linear maps (such as WPRF candidates in **AC0**[**MOD2**]), while availability of AES-hardware acceleration has a significant positive impact on AES-round based instantiations of Diffuse. As an aside, we see our results as informative for ongoing research into developing practical low-depth PRF constructions. In particular,

we demonstrate that low-depth constructions of wPRFs lag considerably behind well-known symmetric primitives that are already offer strong PRF guarantees.

1.1 Related Work

Learning with Rounding. The Learning with Rounding (LWR) assumption [6] is a deterministic variant of the Learning with Errors (LWE) assumption, in which the 'errors' are sampled deterministically based on the choices of $\boldsymbol{a}, \boldsymbol{s} \in \mathbb{Z}_q^n$. In particular, it has been shown that an effective way for demonstrating secure instances of LWR is to compute $\lfloor p/q \cdot \langle \boldsymbol{a}, \boldsymbol{s} \rangle \rceil$, where the choice of p indicates that errors are sampled deterministically in the range $[-q/4p, q/4p)$. Such an approach clearly cannot be adapted to the LPN setting, as $q = 2$.

Learning Parity with Physical Noise The work of [55] shows that LPN-based schemes can be instantiated with 'noise' terms generated by utilising the inexact computation of the inner product. This is formalised as a new assumption called *Learning Parity with Physical-Noise* (see Section 3.1). The effective error rate can be controlled by changing the frequency and voltage overscaling. Due to the practicality of the approach, an LPPN processor has been demonstrated in subsequent works [54,53]. However, cryptanalysis has shown that the errors generated are not always cryptographically secure, which can lead to attacks on the assumption (and subsequent fixes) [11,12]. Note that [55] already postulates the existence of a *deterministic* variant of LPN, but stops short of defining it.

LPN and Low-Depth WPRFs LPN has been studied in relation to construction of weak Pseudorandom Functions (PRFs) in low complexity classes [2,19,23]. A weak PRF relaxes the notion of the standard or "strong" PRFs by having the inputs to the PRF sampled uniformly at random.

Motivated by analogous constructions of weak PRFs based on the Learning with Rounding assumption, [2] considers candidate PRF families of the form $F_A(x) = g(Ax)$ where A is a public matrix and g is essentially a *rounding function*, which adds a "deterministic-noise" to the samples Ax. It constructs g (the so-called Tribes function of Ajtai and Linial [1]) as a specific DNF formula:

$$g(K, x) = \bigvee_{i=1}^{\lambda} \bigwedge_{j=1}^{\log \lambda} x_{ij}. \tag{1}$$

On the negative side, the same work proves that the LPN assumption is not sufficient for proving existence of weak PRFs of the above form in **AC0**[**MOD2**] (constant-depth, with access to MOD2 gates). This candidate was broken in [18].

Boyle et al. in [22] propose a WPRF and a new variant of LPN assumption in relation to it, named *Variable Density-LPN* (or *VDLPN*, for short). For input $x \in \{0,1\}^n$ and key $k \in \{0,1\}^n$, the WPRF is defined as:

$$f_k(x) = \bigoplus_{i=1}^{D} \bigoplus_{j=1}^{w} \bigwedge_{h=1}^{i} (x_{i,j,h} \oplus k_{i,j,h})$$

where D, ω are set to the security parameter λ and $n = w \cdot D \cdot (D-1)/2$. Informally, this candidate weak PRF can be seen as a finite sum of LPN samples with different secret vectors, where each parity term is twice as dense as the previous term ([22], Section 1.3)

Motivated by the previous approaches, [23] proposes a weak and strong PRF candidate in **AC0**[**MOD2**], improving on the design principle of [2]. In particular, they show that generic constructions of strong PRFs can be built from weak PRFs using the formulation $f_{s,K}(x) = \langle x, s \rangle \oplus g(K, x)$, where $s \in \{0,1\}^n$, $K \in \{0,1\}^{(n-1)\times n}$ and $g(K, x)$ is a weak PRF. In [23], the Tribes function is replaced with $g(x) = \bigvee_{i=1}^{\lambda} \bigwedge_{j=1}^{\lambda} \bigvee_{i=1}^{w} x_{ijk}$, which corresponds to a degree-3 representation of the SIPSER function [62,45]. Using this construction, it is possible to evade the attacks shown against Tribes, thanks to higher non-linearity of g. The work also considers a deterministic interpretation of LPN, on which they base their construction of strong PRFs, named *Learning Parity with Simple Deterministic Noise* — we discuss this assumption in Section 3.1.

In [19] the authors propose a construction of low-depth weak PRFs, that is equivalent to a deterministic formulation of LPN, with noise rate 1/3. Their alternative binary weak PRF, $F(k, x) = (\langle k, x \rangle \pmod 2 + \langle k, x \rangle \pmod 3) \pmod 2$, where both key k and input x are binary vectors of length n. The noise term is determined by the value of the inner product $\langle k, x \rangle$ computed over the integers.

PRFs from symmetric encryption. In contrast to the constructions discussed above, block ciphers are very well-understood primitives, that may be modelled as a *pseudorandom permutation* (PRP) to define other cryptographic functionality, e.g. MAC, stream ciphers, etc. Several ways of using a PRP to construct a PRF have also been proposed, some of which discussed in [59], the simplest being truncation of the n-bit PRP output by $m < n$ bits, resulting on a PRF with $(n-m)$-bit outputs. This has been shown in [40] to be secure up to $\approx 2^{\frac{n+m}{2}}$ queries.[6] However, secure instances applying this simple approach will typically result in low-rate PRF constructions. A few more efficient alternatives are discussed in [59], e.g. based on the XOR of PRPs, their sequential computation, input feed-forwarding, as well as combinations of these. The authors then propose a construction with heuristic security – FastPRF – and its instantiation with the AES block cipher, AES-PRF. The choice of AES to instantiate PRF-from-PRP constructions makes sense if targetting real-world deployment in software: as noted in [63], AES-NI instructions can improve performance by up to 10 times when compared with standard AES software implementations.

The *heuristic* approach of FastPRF also features in [7], where the low-latency PRF called Orthros is proposed, as the XOR of two keyed permutations. While each permutation is not claimed to be strong enough as stand-alone block ciphers (i.e. they do not claim to behave like a PRP), they may however be combined to define a secure PRF. Other proposals of low-latency PRF based on keyed permutations include the Kirby construction [56], and its instantiation Koala [34].

[6] In the case where there is no truncation, i.e. $m = 0$, this gives the standard birthday bound for the well-known PRP-PRF switch technique.

In this work, we employ some of these approaches to define our deterministic noise functions. We may use AES-based keyed permutations, of which we may not be able to claim PRP-security, but when combined with truncation and folding operations, appear to lead to secure DLPN constructions.

1.2 Paper outline

This work is organised as follows. We begin in section 2 by specifying the notation and necessary definitions. In section 3 we define the Deterministic LPN problem, and consider the relation with standard LPN and its main variants. We discuss in section 4 how to instantiate the noise function and provide concrete instantiations that are expected to have good software performance. We provide arguments for the security of our proposed constructions in section 5, and report the experimental results for their benchmarks in section 6.

2 Preliminaries

2.1 Notation

Matrices (e.g. $\boldsymbol{M}$) and vectors (e.g. $\boldsymbol{v}$) will be denoted by upper- and lower-case bold font, respectively. We may interchangeably refer to ℓ-bit strings by their counterpart vectors in $\mathbb{Z}_2^\ell$. We will denote by $[n]$ the set $\{1, \ldots, n\}$.

We denote by Ber_τ an efficient Bernoulli sampler with probability parameter τ. We use PPT to refer to polynomial probabilistic time, in the context of discussing the efficiency of some algorithm A. For a set X, we use $x \leftarrow\!\$\ X$ to denote sampling uniformly from it.

We use λ to denote a security parameter, and let $\mathsf{negl}(\lambda)$ denote a negligible function. We say that some problem P is computationally hard to solve, if for all PPT algorithms $\mathcal{A}$, we have that $\Pr[\mathcal{A}$ succeeds in solving $P] < \mathsf{negl}(\lambda)$.

2.2 Learning Parity with Noise

The Learning Parity with Noise (LPN) problem is one of the most prominent hard learning problems with application in cryptography [15]. Informally, it asks one to distinguish noisy inner products in $\mathbb{Z}_2$ from uniformly sampled bits.

Definition 1 (Decisional/Search LPN problem). *Let $\boldsymbol{A} \leftarrow\!\$\ \mathbb{Z}_2^{m\times\ell}$, for polynomials $m = m(\lambda)$ and $\ell = \ell(\lambda)$. Then, for $0 < \tau < 1/2$, consider the following distributions: (i) D_0: sample $\boldsymbol{s} \leftarrow\!\$\ \mathbb{Z}_2^\ell$ and $\boldsymbol{e} \leftarrow\!\$\ (\mathsf{Ber}_\tau)^m$; output $(\boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e})$; and (ii) D_1: output $\boldsymbol{u} \leftarrow\!\$\ \mathbb{Z}_2^m$. The* decisional Learning Parity with Noise *(LPN) problem considers a PPT algorithm $\mathcal{A}$, which is given the input $(\boldsymbol{A}, D_b)$ for $b \leftarrow\!\$\ \mathbb{Z}_2$, outputs $b' \in \{0,1\}$, and succeeds if $b' = b$. Likewise, the* search LPN *problem requires a PPT algorithm $\mathcal{A}'$ to, given an output from D_0, recover the vector $\boldsymbol{s}$.*

The success probability for the algorithm $\mathcal{A}$, is given by:

$$p^{\mathsf{LPN}}_{\mathcal{A},m,\ell,\tau}(\lambda) = \Pr\left[b' = b \middle| \begin{smallmatrix} b \leftarrow\$ \{0,1\} \\ b' \leftarrow \mathcal{A}((\boldsymbol{A}, D_b)) \end{smallmatrix}\right],$$

and corresponding advantage by: $a^{\mathsf{LPN}}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{LPN}}_{\mathcal{A},m,\ell,\tau}(\lambda) = |p^{\mathsf{LPN}}_{\mathcal{A},m,\ell,\tau}(\lambda) - 1/2|$. The *decisional LPN assumption* states that solving $\mathsf{LPN}_{m,\ell,\tau}$ is hard (i.e. $a^{\mathsf{LPN}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$) for any PPT adversary algorithm $\mathcal{A}$. The hardness of search LPN is defined equivalently. In this work we will refer always to the decisional variant of LPN, which is known to be equivalent to the search variant [15].

2.3 Pseudorandom and Unbalanced Functions

Definition 2 (Weak/Strong PRFs). *Let $\mathcal{K}, \mathcal{X}, \mathcal{Y}$ be the key, input, and output spaces, respectively, parameterised by a security parameter λ. Let $\mathcal{F} = \{f_k \mid f_k : \mathcal{X} \to \mathcal{Y}\}$ be a family of functions indexed by $k \in \mathcal{K}$, and $\mathcal{RF}$ be the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.*
Then, for $\{x_i\}_{i\in[m]}$ (for some $m = \mathsf{poly}(\lambda)$), let D_0 be the distribution that samples $k \leftarrow\$ \mathcal{K}$, and outputs $\{f_k(x_i)\}_{i\in[m]}$, and let D_1 be a distribution that samples $f \leftarrow\$ \mathcal{RF}$, and outputs $\{f(x_i)\}_{i\in[m]}$.

- *We say that $\mathcal{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a (strong)* pseudorandom function *(PRF) family, if all PPT algorithms $\mathcal{A}$ attempting to distinguish between D_0 and D_1, where $\{x_i\} \leftarrow \mathcal{A}$, have advantage bounded by $\mathsf{negl}(\lambda)$.*
- *We say that $\mathcal{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a* weak pseudorandom function *(WPRF) family, if all PPT algorithms $\mathcal{A}$ attempting to distinguish between D_0 and D_1, where $\{x_i\} \leftarrow\$ \mathcal{X}$, have advantage bounded by $\mathsf{negl}(\lambda)$.*

We state the notion of τ-unbalanced function families, defined in [23].

Definition 3 (τ-Unbalanced Function Families [23]). *We say that $\mathcal{F}_\tau = \{f \mid f : \mathcal{X} \to \{0,1\}\}$ is a τ-Unbalanced Function Family if for $f \leftarrow\$ \mathcal{F}_\tau$, we have $\Pr_x[f(x) = 1] = \tau$.*

While the notion of τ-unbalanced function families is sufficiently generic, we will also work with a more concrete definition that explicitly takes into account indistinguishability from sources of randomness. In particular, we consider the notion of an unbalanced weak pseudorandom function in Definition 4.

Definition 4 (τ-Unbalanced Weak Pseudorandom Function). *Let $\ell = \ell(\lambda)$, $m = m(\lambda)$, $n = n(\lambda) \in \mathbb{N}$, let $\mathcal{F}_\tau$ be a τ-unbalanced function family (Definition 3), and let $g_\tau : \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}$.*
Now, let $\{\boldsymbol{x}_i\}_{i\in[m]} \leftarrow\$ (\{0,1\}^\ell)^m$ be a set of m independently sampled binary vectors of length n. If D_0 denotes the distribution that samples $\boldsymbol{s} \leftarrow\$ \{0,1\}^n$, and then outputs $\{g_\tau(\boldsymbol{x}_i, \boldsymbol{s})\}_{i\in[m]}$, and D_1 the distribution that samples $f \leftarrow\$ F_\tau$, and outputs $\{f(\boldsymbol{x}_i)\}_{i\in[m]}$. We say that g is a τ-unbalanced weak pseudorandom function (τ-UWPRF) for $\mathcal{X}$ if, for all PPT algorithms $\mathcal{A}$, we have

$$\Pr\left[b' = b \middle| \begin{smallmatrix} b \leftarrow\$ \{0,1\} \\ b' \leftarrow \mathcal{A}((\{x_i\}_{i\in[m]}, D_b)) \end{smallmatrix}\right] = 1/2 + \mathsf{negl}(\lambda).$$

Finally we restate in Lemma 1 a result that was implied in [23], but only proven in the paper's (non-public) full version [24].

Lemma 1 (τ-UWPRF to WPRF [24]). *Let g_τ be a τ-UWPRF. Then, under the assumption that decisional $\mathsf{LPN}_{m,\ell,\tau}$ is hard, we can obtain a standard WPRF (Definition 2).*

Proof. Assume that $\ell = n$, and thus $g_\tau : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$, and let $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be defined as $L(\boldsymbol{x}, \boldsymbol{s}) = (\langle \boldsymbol{x}, \boldsymbol{s} \rangle + g_\tau(\boldsymbol{x}, \boldsymbol{s}))$. We now construct an adversary $\mathcal{B}$ that attempts to distinguish between $(\{\boldsymbol{x}_i\}_{i\in[m]}, L(\boldsymbol{x}_i))$ and $(\boldsymbol{x}_i, u \leftarrow\!\$\ \{0,1\})$. If no such $\mathcal{B}$ exists with advantage greater than $\mathsf{negl}(\lambda)$, then we conclude that L is a WPRF.

- h_1: We replace g_τ with a randomly sampled function $f : \{0,1\}^\ell \mapsto \{0,1\}$ from a τ-unbalanced function family F_τ. This follows directly from Definition 4.
- h_2: We replace outputs of F_τ with independent samples from a Bernoulli distribution with parameter τ (Ber_τ). These games are statistically indistinguishable, since the only chance of distinguishing the two arises when we have a collision $x_i = x_j$ for $i \neq j$ and $i, j \in [m]$ for the input samples.
- h_3 We now replace the entire output of L, with $u \leftarrow\!\$\ \{0,1\}$. Note that this follows because $L(\boldsymbol{x}_i, \boldsymbol{s}) = \langle \boldsymbol{x}_i, \boldsymbol{s} \rangle + \mathsf{Ber}_\tau$, which is exactly the form of $\mathsf{LPN}_{m,\ell,\tau}$.

After transitioning to h_3, we have completed the proof.

3 Deterministic Variants of LPN

In order to use instances of LPN in applications, we must consider the following necessary computation. Given a uniformly sampled secret $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ and m samples of random vectors $\boldsymbol{a}_i \in \mathbb{Z}_2^\ell$, we will compute $(\langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i)_{i=1}^m$ for error bits $\{e_i\}_{i\in[m]}$ sampled independently from Ber_τ. As noted, sampling of randomness is a non-trivial task for many real-world devices.

The same may also apply for the parameters a_i (public) and s (secret) above, but for example in LPN-based cryptographic schemes, s can be viewed as a secret key, while a_i's can be provided from public sources of randomness (such as randomness beacons), or by a Verifier device in identification schemes like the HB protocol [50] and follow up works.

In the case of LWE [61], deterministic variants that allow "sampling" errors based alone on the long-term public and secret data have been developed and studied. Similar observations have been made for LPN [55,11,19,23], but the exact nature of the assumption has often been left implicit, or only concretely specified for specific parameterisations. In the following sequence of definitions, we give a new asymptotic formulation of (decisional) deterministic LPN that unifies prior art, and upon which we can then use to postulate hard variants of the problem. Note that the search variant of deterministic LPN follows naturally, and deterministic Ring LPN [48] may also be defined using the same framework.

Definition 5 (Decisional Deterministic LPN). *Let $m = m(\lambda)$, $\ell = \ell(\lambda) \in \mathbb{N}$, and $0 < \tau < 1/2$. Let $g_\tau : \mathbb{Z}_2^{m\times\ell} \times \mathbb{Z}_2^{\ell} \mapsto \mathbb{Z}_2^m$ be a function. Let $\boldsymbol{A} \leftarrow\!\!\$\ \mathbb{Z}_2^{m\times\ell}$, and consider the following distributions: (i) D_0: sample $\boldsymbol{s} \leftarrow\!\!\$\ \mathbb{Z}_2^{\ell}$; output $(\boldsymbol{A}\cdot\boldsymbol{s} + g_\tau(\boldsymbol{A},\boldsymbol{s}))$; and (ii) D_1: output $\boldsymbol{u} \leftarrow\!\!\$\ \mathbb{Z}_2^m$. Then, the* deterministic LPN *problem (denoted $\mathsf{DLPN}_{m,\ell,g_\tau}$) considers a PPT algorithm $\mathcal{A}$, that is given the input $(\boldsymbol{A}, D_b)$ for $b \leftarrow\!\!\$\ \mathbb{Z}_2$, outputs $b' \leftarrow\!\!\$\ \mathbb{Z}_2$, and succeeds if $b' = b$.*

For the adversary algorithm $\mathcal{A}$, we denote their success probability by:

$$p^{\mathsf{DLPN}}_{\mathcal{A},m,\ell,g_\tau}(\lambda) = \Pr\left[b' = b \middle| \begin{smallmatrix} b\leftarrow\!\!\$\{0,1\} \\ b'\leftarrow\mathcal{A}((\boldsymbol{A},D_b)) \end{smallmatrix}\right],$$

and corresponding advantage by: $a^{\mathsf{DLPN}}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{DLPN}}_{\mathcal{A},m,\ell,g_\tau}(\lambda) = |p^{\mathsf{DLPN}}_{\mathcal{A},m,\ell,g_\tau}(\lambda) - 1/2|$. The *deterministic LPN assumption* states that solving $\mathsf{DLPN}_{m,\ell,g_\tau}$ is hard (i.e. $a^{\mathsf{DLPN}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$) for any PPT adversary algorithm $\mathcal{A}$.

Remark 1 (Number of samples). Note that the parameter m defines the number of samples that the adversary sees. Therefore, we may abuse notation and simply write $\mathsf{DLPN}_{\ell,g_\tau}$, when we consider m to be arbitrary.

Remark 2 (On families of functions). We could write Definition 5 with respect to a function family $G_\tau = G_\tau(\lambda)$, from which g_τ is then sampled. We could then demonstrate hard instances of DLPN corresponding to different function families G_τ. However, in the following we will focus on specific instances of the function g_τ, and therefore prefer to consider this formulation.

3.1 Relationships with LPN and Variants

Below, we state Lemma 2 which shows the equivalence of $\mathsf{LPN}_{m,\ell,\tau}$ and $\mathsf{DLPN}_{m,\ell,g_\tau}$ under the assumption that τ-UWPRF exist. We note that the result is valid for both the search and decisional variants.

Lemma 2 ($\mathsf{LPN} \approx_c \mathsf{DLPN}$). *Let $\mathsf{LPN}_{m,\ell,\tau}$ be the decisional LPN problem. For some function, let $g_\tau : \{0,1\}^{m\times\ell} \times \{0,1\}^{\ell} \to \{0,1\}^m$ be an unbalanced weak pseudorandom function, and let $\mathsf{DLPN}_{m,\ell,g_\tau}$ be the decisional deterministic LPN problem, as given in Definition 5. Then, if g_τ is an unbalanced weak pseudorandom function (Definition 4), the two problems are computationally equivalent.*

Proof. The proof follows almost trivially from the proof of Lemma 1. Consider an instance of the LPN problem $\boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e}$ for $\boldsymbol{e} \leftarrow\!\!\$\ \{0,1\}^m_\tau$. Then, we can apply the hybrid arguments in reverse to transition from h_3 to h_1, where $\boldsymbol{e} \leftarrow g_\tau(\boldsymbol{A},\boldsymbol{s})$ is generated as the result of the τ-UWPRF, g_τ. Note that in h_1, we have exactly an instance of $\mathsf{DLPN}_{m,\ell,g_\tau}$. We can perform the reverse transformation to guarantee computational equivalence.

Learning Parity with Physical Noise. This LPN variant was proposed in [55], where the LPN function is replaced by the a physical device (modelled by a function $\mathsf{PF} : \mathbb{Z}_2^{\ell} \times \mathbb{Z}_2^{\ell} \to \mathbb{Z}_2$), that outputs inexact inner products with some probability ϵ.

Definition 6 (Learning Parity with Physical Noise (LPPN) [55,11]). *Let* $\mathsf{PF}_\epsilon : \mathbb{Z}_2^\ell \times \mathbb{Z}_2^\ell \times \{0,1\}^* \to \mathbb{Z}_2$ *be a function such that* $\mathsf{PF}_\epsilon(\boldsymbol{a}, \boldsymbol{b}; r) = \langle \boldsymbol{a}, \boldsymbol{b} \rangle + e_r$ *for some* $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_2^\ell$*, randomness* $r \leftarrow\!\!\$\ \{0,1\}^*$*, and* $e_r \leftarrow\!\!\$\ \{0,1\}$ *such that* $\Pr_r[e_r = 1] = \epsilon \in [0, 1/2)$*. Then we say that the* $\mathsf{LPPN}_{1,\ell,\epsilon}$ *is hard if the output of* PF_ϵ *(on uniformly distributed randomness* r*) is indistinguishable from a uniformly random value in* $\mathbb{Z}_2$*.*

Note that the LPPN problem is equivalent to the standard LPN problem in the general case. However, typically the inner product function PF_ϵ [12,11] is implemented as a physical circuit, where noise is introduced through natural perturbations that occur in the evaluation of the function. In particular, we cannot guarantee that the probability density function associated with $\Pr_r[e_r = 1]$ outputs errors with sufficient frequency or unpredictably to generate secure LPN samples. Therefore, the security guarantee is conjectured on the specific physical function used, and does not guarantee LPN-hard noise rates. Furthermore, the initial noise rate itself depends on the physical phenomena that is occurring, and thus the distribution of e_r is somewhat unpredictable and independent of the computation performed. As such, we cannot use this approach to directly generate LPN samples where noise rates need to be sampled with specific values.

Learning Parity with Simple Deterministic Noise In [23], a variant of DLPN is briefly discussed, known as *Learning Parity with Simple Deterministic Noise*. We detail the form of this problem in Definition 7 below.

Definition 7 (Learning Parity with Simple Deterministic Noise (LPSDN)). *Let* ℓ, τ *all be defined as in Definition 5. Let* $f : \mathcal{K} \times \mathbb{Z}_2^{m \times \ell} \mapsto \mathbb{Z}_2^m$ *be a* τ*-UWPRF, where* $\mathcal{K}$ *is some arbitrary key space (e.g.* $\{0,1\}^\lambda$*). Let* $\boldsymbol{A} \leftarrow\!\!\$\ \mathbb{Z}_2^{m \times \ell}$*,* $k \leftarrow\!\!\$\ \mathcal{K}$*, and consider the following distributions:*

1. D_0*: sample* $\boldsymbol{s} \leftarrow\!\!\$\ \mathbb{Z}_2^\ell$*; output* $(\boldsymbol{A} \cdot \boldsymbol{s} + f(k, \boldsymbol{A}))$*;*
2. D_1*: output* $\boldsymbol{u} \leftarrow\!\!\$\ \mathbb{Z}_2^m$*.*

Then, solving the $\mathsf{LPSDN}_{m,\ell,\tau}$ *problem requires distinguishing between* D_0 *and* D_1 *with non-negligible probability.*

It is clear that $\mathsf{LPSDN}_{m,\ell,\tau}$ and $\mathsf{DLPN}_{m,\ell,g_\tau}$ are almost equivalent. The main difference is that in the latter, we require that $\mathcal{K} = \mathbb{Z}_2^\ell$, and $k = \boldsymbol{s}$. This ensures a true deterministic link between the error vector, and the choices of $\boldsymbol{A}$ and $\boldsymbol{s}$, and simplifies the exposition of the problem – in practical applications, it also means that we only have a single secret to store, instead of a secret and a key.

4 Concrete Constructions of Unbalanced Functions

The recent surge in the number of practical protocols and constructions built from LPN [13,27,21] illustrates the potential for using the problem as a security basis in various applications. In particular, LPN's role in building practical zero-knowledge proof systems with low prover overheads (based on VOLE) provides motivation for instantiating such primitives in low-power settings, where sampling secure randomness may not be guaranteed.

In Section 3, we showed that $\mathsf{DLPN}_{m,\ell,g_\tau}$ is essentially equivalent to $\mathsf{LPN}_{m,\ell,\tau}$, when g_τ is a τ-UWPRF. Therefore, our primary question is: how to construct g_τ in a secure manner? To address this question, we will consider two functions $\mathsf{Diffuse} : \mathbb{Z}_2^{m\times\ell} \times \mathbb{Z}_2^{\ell} \to \mathbb{Z}_2^n$ and $\mathsf{Folder}_\tau : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$, for some polynomial $n = n(\lambda)$ and $\tau \in (0, 1/2)$. We will denote these functions by D and F_τ, respectively. Then, we will build g_τ as the composition $g_\tau = \mathsf{F}_\tau \circ \mathsf{D}$. Concretely, this approach will allow us to separate the pieces of the function that manage the *pseudorandomness* of the WPRF, from those that ensure that the output is τ-*unbalanced*.

To illustrate our approach, we start with a simple example based on well-known cryptographic primitives. We then discuss general Boolean formulae that appear amenable to building τ-unbalanced functions (Folder_τ), and end the discussion by focusing on the pseudorandomness of $\mathsf{Diffuse}$. In particular, we discuss alternative, practical approaches for providing security in the LPN setting. In some cases, such constructions are *conjectured* secure, based on the supposed hardness of less-standard cryptographic objects (such as prior work on WPRFs [19,23], or low-depth symmetric primitives). In the case where we specify new constructions, we provide justification for their hardness in Section 5.

Simple, Secure Candidate from PRFs. Let $\mathsf{PRF} : \mathcal{K} \times \mathcal{X} \to \mathbb{Z}_2^n$ be a pseudorandom function for $\mathcal{K} = \mathbb{Z}_2^{m\times\ell}$ and $\mathcal{X} = \mathbb{Z}_2^{\ell}$. Then, we can instantiate a secure instance of g_τ for any $\tau = 2^{-\mu}$ for $\mu \leq n/m$. To see this, consider the function $\mathsf{D} = \mathsf{PRF}$, and F_τ defined as (where $\mu = n/m$, for simplicity):

$$\mathsf{F}_\tau(y \in \mathbb{Z}_2^n) = \bigwedge_{i=1}^{\mu} y_i \| \ldots \| \bigwedge_{i=1}^{\mu} y_{((m-1)\cdot n/m)+i}. \tag{2}$$

Function F_τ simply computes a sequential AND of the first μ bits. Considering $g_\tau = \mathsf{F}_\tau \circ \mathsf{D}$, Lemma 3 shows that it achieves the necessary security guarantee.

Lemma 3. *Let* D *and* F_τ *be defined as above, then* $g_\tau = \mathsf{F}_\tau \circ \mathsf{D}$ *is a* τ*-unbalanced function.*

Proof. The proof proceeds as a series of hybrid steps, as detailed below.

- h_1: Replace D with a random function $D : \mathcal{X} \mapsto \mathbb{Z}_2^n$. This follows directly from the fact that D is a PRF.
- h_2: Treating the output of $D(x)$ as a sequence of random bits $y \in \{0,1\}^n$, consider the first bit output by $g_\tau(y) = \mathsf{F}_\tau(D(x)) = \bigwedge_{i=1}^{\mu} y_i$. If any of the μ bits of y in question are set to 0, then $F(y) = 0$, and otherwise the first bit of $F(y)$ is 1. The probability that the first bit is 1 is therefore exactly equal to $2^{-\mu}$. Therefore, we can transition to a game where we replace the first bit of the output of g_τ with the output of Ber_{2^μ}. In general, we can make m hybrid transitions, one for each bit of the desired output of g_τ, to eventually replace g_τ with m independent samples from Ber_τ. Therefore, for any choice of $\mu \leq n/m$, then g_τ represents a $(2^{-\mu})$-unbalanced PRF.

Remark 3. Recall that, for establishing computational equivalence of DLPN and LPN (Lemma 2) we require that g_τ be τ-UWPRF (Definition 4). While Definition 4 is a rather bespoke notion, concrete instantiations of such a function

remain elusive. We provide one possible concrete instantiation, which is the composition $g_\tau = \mathsf{F}_\tau \circ \mathsf{D}$. Now, if we were to choose D as a (strong) PRF in this formulation, then this would clearly allow us to prove that this composition is a τ-UWPRF in Lemma 3. However, such strong PRFs are typically onerous from a performance perspective. Therefore, one must rely on the principle that secure instantiations of DLPN arise as long as g_τ is a τ-UWPRF. This would then enable us to obtain computational equivalence of DLPN with LPN. In summary, our approach provides a flexible framework for DLPN instantiations, where one can replace D with a suitable function, and suitability is determined as an application-specific metric. In our experimental evaluation in Section 6, our focus is on instantiating D with as high software performance as possible. This leads us naturally to τ-UWPRF candidates that perform well in this context.

With this simple candidate, there is a temptation to suggest that the problem is solved. However, many applications of LPN require careful management of the noise rate reaching specific capabilities, while ensuring meaningful security. In particular, it is common to configure LPN noise rates to all values of 2 decimal places in precision, between 0.49 and 0.01 (see [35] for various parameterisations and their effective security). Therefore we would like to be able to devise instantiations of τ-UWPRFs with arbitrary choices of $\tau \in (0, 1/2)$.

4.1 Folding for Arbitrary Noise Rates

Before constructing Folder_τ for arbitrary τ, we highlight recent work towards the building low-depth WPRFs from Boolean formulae. We will eventually build our own configurable Boolean formulae that allows us to simulate τ-UWPRFs for arbitrary τ, where previous instantiations provide suitable parameterisations for only fixed values. For now, we continue to assume that $\mathsf{Diffuse}$ is instantiated as a pseudorandom function, which is important for establishing security and functionality. We note that our constructions and previous attempts mentioned in this section fall into the case of $m = 1$ in Definition 4.

Recall, the Tribes and SIPSER constructions discussed in Section 1.1. It can be shown that $\Pr_{x\in\{0,1\}^n}[g(x) = 1] = 1-(1-\frac{1}{2^{\log \lambda}})^\lambda$ and $1-\left[1-(\frac{1}{2^w})^\lambda\right]^\lambda$ when g is either the Tribes or the SIPSER function. We note that, for a desired security parameter, it is not possible to configure the unbalancedness of this function (if we consider the function as a τ-UWPRF). In other words, a concrete choice of τ is implied by the concrete choice of λ. We address this problem by providing a construction of Folder_τ that can be instantiated for any value of $\tau \in (0, 1/2)$

Constructing configurable instances. Essentially we are dealing with problem of constructing *unbalanced* Boolean function. The problem of constructing Boolean functions with various desired properties has been widely studied, but the focus is mostly on balanced Boolean functions (for instance, see [25]). To the best of our knowledge, there is limited literature for construction of unbalanced Boolean functions for given value of n and $\tau \in (0, 1/2)$.

In the truth-table representation of the function, we would like to construct a τ-UWPRF for any $\tau \in (0, 1/2)$ with corresponding truth table of hamming

weight $\approx \lfloor \tau 2^n \rfloor$. However, we would like an algebraic representation of the function for theoretical analysis and also for large values of n, representing the function as a truth table would appear to be impractical. With this in mind, we present a more flexible approach of constructing depth-2 unbalanced functions in DNF formula using a function description that follows a similar format to that of the Tribes function [2]. Here, the advantage of lies in high efficiency, for example due to parallelizable ANDs. when instantiating the function for use in real-world applications. This was also the primary goal of prior work of [2,23]

Arbitrary noise rates from *universal* Tribes. Consider the Tribes Boolean function [1] as the folding function. As in its original form, we can interpret the Tribes function differently from [2] for parameters ω, s such that:

$$\mathsf{Tribes}_{\omega,s}(x_1, \ldots, x_n) = \bigvee_{i=1}^{s} \bigwedge_{j=1}^{\omega} x_{i,\omega}$$

where $n = \omega \times s$.[7] Here we view n-bit inputs as divided into s blocks or "tribes" where each tribe consists of ω bits. The function then is the OR of s ANDs of ω bits. We have $\Pr_{x \in \{0,1\}^n}[\mathsf{Tribes}_{\omega,s}(x) = 1] = 1 - (1 - \frac{1}{2^\omega})^s$, the "unbalancedness"

We can generalise this formulation more, by choosing variable tribe sizes to get the desired unbalancedness, while using only the first few bits of the input. The reader may wonder how it is we plan to guarantee security in this regime, given the aforementioned attacks [18]. For now, we assume that the n bits we receive as input from the function Diffuse are pseudorandom and are complex functions of input LPN samples. This is a departure from the original Tribes map, that received inputs from a linear map. However, we highlight later (in Section 4.2) how we can produce pseudorandom (or at least non-linear, balanced) output very efficiently in practice. We define this general formulation as *universal* description for Tribes (which we denote UTribes), and describe it below. The idea is that, we sequentially choose the 'next' best tribe size, until sufficiently low approximation of τ is reached, or we run out of available bits to define the tribe sizes. Formally, we denote out construction by $\mathsf{UTribes}_{\omega_1,\ldots,\omega_m}$, where $\omega_1, \ldots, \omega_t$ are the respective Tribes sizes, and the formal statement of the construction in Definition 8

Definition 8. *For $\tau \in (0, 1/2)$, and let $\omega_0, \ldots, \omega_t \in \mathbb{N}$. Then $\mathsf{UTribes}_{\omega_1,\ldots,\omega_t} : \{0,1\}^n \to \{0,1\}$ is a function defined as follows:*

$$\mathsf{UTribes}_{\omega_1,\ldots,\omega_t}(x_1, \ldots, x_n) = \bigvee_{i=1}^{t} \bigwedge_{j=1}^{\omega_i} x_{\omega_i}.$$

Proposition 1. *Let $\tau \in (0, 1/2)$, and let $\omega_0 \in \mathbb{N}$ be the smallest positive integer such that $\frac{1}{2^{\omega_0}} < \tau$. Define the Boolean function $\mathsf{UTribes}_{\omega_1,\ldots,\omega_t}$ as in Definition 8. First, define $f^0 : \{0,1\}^n \to \{0,1\}$ as follows:*

$$f^0(x_1, \ldots, x_n) = x_1 \wedge \ldots \wedge x_{\omega_0}$$

[7] This is sometimes known as a generalised form of Tribes [42].

Inductively define ω_t for $t > 0$ as follows. Suppose we have $\omega_0, \ldots, \omega_t$ and let

$$\Omega_t = \sum_{i=0}^{t} \omega_i, \;\; \tau_t = \Pr_{x \in \mathbb{F}_2^n}[f^t(x) = 1]$$

Then define $\omega_{t+1} \in \mathbb{N}$ as the smallest positive integer larger than ω_t such that $\frac{1}{2^{\omega_{t+1}}} < (\tau - \tau_t)$ and $\Omega_{t+1} < n$. Define

$$f^{t+1}(x) = f^t(x_1, \ldots, x_n) \bigvee \left(x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}\right)$$

Then for $t \in \mathbb{N}$, if $\epsilon_t = \tau - \tau_t$ is the approximation error, we have that $\epsilon_0 > \epsilon_1 > \ldots \epsilon_t$ and $\epsilon_t < 2^{-(\omega_{t+1}-1)}$. In other words, with increasing t, $NR(f_t)$ gives an increasingly better approximation of τ.

Proof. Since f_{m+1} is an OR between two terms, f^t and the next 'tribe' terms $\left(x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}\right)$, we have that

$$\begin{aligned}
\Pr[f^{t+1}(x) = 1] &= \Pr[f^t(x) \vee (x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}) = 1] \\
&= \Pr[f^t(x) = 1] + \Pr[(x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}) = 1] \\
&\quad - \Pr[f^t(x) = 1, (x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}) = 1] \\
&= \Pr[f^t(x) = 1] + \Pr[(x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}) = 1] \\
&\quad - \Pr[f^t(x) = 1] \times \Pr[(x_{\Omega_t+1} \wedge \ldots \wedge x_{\Omega_t+\omega_{t+1}}) = 1] \\
&= \tau_t + \frac{1}{2^{\omega_{t+1}}}(1 - \tau_t) \\
\Pr[f^{t+1}(x) = 1] &< \tau_t + \frac{1}{2^{\omega_{t+1}}} \qquad (3)
\end{aligned}$$

Since ω_{t+1} was chosen such that $\frac{1}{2^{\omega_{t+1}}} < (\tau - \tau_t)$ we have, $\Pr[f^{t+1}(x) = 1] = \tau_{t+1} < \tau$. Similarly we have that $\tau_{t+1} = \Pr[f^{t+1}(x) = 1] = \tau_t + \frac{1}{2^{\omega_{t+1}}}(1 - \tau_t)$, which implies $\tau_{t+1} > \tau_t$.

We have therefore shown that starting with $\tau_0 < \tau$ we have a strictly increasing sequence of $\tau_0 < \tau_1 < \ldots < \tau_t < \tau_{t+1} < \tau$ and hence a decreasing sequence $\epsilon_0 > \epsilon_1 > \ldots \epsilon_t$ where $\epsilon_t = \tau - \tau_t$ is the approximation error.

Finally, recall that ω_{t+1} was chosen such that $\frac{1}{2^k} > (\tau - \tau_t)$ for all positive integers $k < \omega_{t+1}$ and $\frac{1}{2^{\omega_{t+1}}} < (\tau - \tau_t)$, So, we have that

$$\epsilon_t = (\tau - \tau_t) < \frac{1}{2^{\omega_{t+1}-1}} \qquad (4)$$

We give examples on how to realise g_τ, for different values of τ in Appendix B

4.2 Instantiating the Diffuse Function

We discuss in this section the instantiation of the function Diffuse (D), whose output is provided as input to the folding function (F_τ). From a security point

of view, to allow us to apply the guarantees from Lemma 3, we need D to provide *pseudorandomness*. Thus, any pseudorandom function that can process long enough inputs, and with enough output bits, could be viable for this use-case. Recall, however, that the D function need only satisfy weak pseudorandomness, since the input to D is assumed to be randomly sampled; moreover an adversary does not see the output of D, but only the one bit output of $\mathsf{F}_\tau \circ \mathsf{D}$.

Given our motivation of sampling LPN error in low-capability devices, we also require that such functions are highly efficient. Thus, rather than relying on low-depth theoretical complexity classes, we will consider performance benchmarking as a metric of efficiency instead. We recall, as stated in Remark 3, that D can be replaced with any other function suitable according to the metric for application in question. Here, our metric is software performance, not low depth. In the following, we discuss our suggested constructions to be used in place of D, and to be evaluated in our experiments in Section 6.

ASCON. Ascon-PRF [30] is a cryptographic primitive from the Ascon v1.2 family [29], designed to provide efficient pseudorandom functionality for resource-constrained devices. It uses the same Ascon permutation as selected by NIST as the standard for lightweight cryptography [66]. We will use the AsconPRFs with arbitrary input size and 16/32 byte output.

Candidate Low-Depth Weak PRFs. As discussed early, the literature on low-depth WPRF constructions has advanced over the past few years. In particular, the candidates from [23,19] appear to have avoided polynomial-time attacks [26]. However, given that the construction of [19] only allows realising error rates of $1/3$ (or converging to $1/2$ in the general case, by altering the moduli), we will consider the SIPSER-based construction of [23]. This assumes that we can achieve approximations of arbitrary τ, which is not a given due to the lack of flexibility in defining the noise rate in the standard SIPSER formulation. That said, we may define for now $\mathsf{D}_{\boldsymbol{M}}(\boldsymbol{x}) = \boldsymbol{M} \cdot \boldsymbol{x}$, where $\boldsymbol{M} \leftarrow\!\!\$\ \mathbb{Z}_2^{\ell \times \ell}$.

A problem with the above construction, when interpreted within our deterministic LPN framework is that $\boldsymbol{M}$ is an extra secret parameter. To get around this and make the formulation compatible, we can instead sample $\boldsymbol{M} \leftarrow \mathsf{PRG}(\boldsymbol{A} \| \boldsymbol{s})$. We note that if we define $g_\tau = \mathsf{UTribes}_{\omega_1,\ldots,\omega_t}(\mathsf{PRG}(\boldsymbol{A}\|\boldsymbol{s}) \cdot \boldsymbol{x})$, then we may fall foul of the same cryptanalysis that impacted the construction of [2]; see [18]. While the construction of [2] used a public matrix, and our matrix can be kept secret, we note that our universal Tribes formulation is not meaningfully different from the original Tribes function, from a cryptanalytic perspective. Therefore, while SIPSER does not maintain the same level of configurability as $\mathsf{UTribes}$, we believe that using SIPSER here may be necessary from a security perspective. However, we also note that defining g_τ as above at least maintains a performance lower bound for this approach, since $\mathsf{UTribes}$ is a more efficient DNF formula than SIPSER.

Simpira. While PRFs in **AC0**[**MOD2**] may lead to fast implementations by their very nature, this ignores the fact that modern computing architectures are equipped with specific hardware instructions for computing AES rounds, leading to very efficient implementations of AES-based designs. From a practical

perspective, this corresponds to considering low-depth PRFs in **AC0**[**AES**], i.e. constant-depth circuits with gates for computing AES rounds. Note that this question has already motivated research on constructing various symmetric key primitives based on the AES-round function, making use of AES-NI for efficient implementation [33,8,51,17]. Our first candidate based on this approach is Simpira (version 2) [43], a family of permutations using the AES round function as a building block, supporting inputs of $128 \times b$ bits, where b is a positive integer.

LightMAC. Following this approach even further, we propose a second candidate construction based on LightMAC [57]. LightMAC is a parallelisable MAC construction based on a block cipher mode of operation, with proof of security based on the underlying PRP. The original work instantiates LightMAC with PRESENT and AES [57]. In our case, to build D aiming for higher efficiency, we do not use the full AES, but rather instantiate it with **AES4**, a reduced-round version of AES with 4 rounds. **AES4** is obviously not a PRP-secure, but *may* be used securely as a building block of a mode of operation when not aiming for the standard block cipher functionality. **AES4** has already been used in the AE scheme AEZ [49], and in fact in EliMAC [31], a very similar construction to LightMAC. However, to prevent length-extension attacks in the MAC setting, EliMAC finishes the tag computation with a full 10-round AES operation. We conjecture that this is an overkill in our setting, where we only require pseudorandom output bits, and the attacker doesn't see the entire output of $\mathsf{Diffuse}$. Thus our choice to use LightMAC instantiated with **AES4**.

5 Security Analysis

From the symmetric-key cryptanalysis point of view, an instance of the determinist LPN construction is essentially a Boolean function defined as the sum of a secret linear function (the inner product by $\boldsymbol{s}$) and a secret non-linear function g_τ with *known* structure. This Boolean function takes as input known, but randomly sampled vectors $\boldsymbol{x}_i$, and produces as output $b_i \in \mathbb{Z}_2$.

$$b_i = \langle \boldsymbol{s}, \boldsymbol{x}_i \rangle + g_\tau(\boldsymbol{s}, \boldsymbol{x}_i).$$

The adversary has access to $(\boldsymbol{x}_i, b_i)$ and wishes to recover $\boldsymbol{s}$ (in the search variant of DLPN). In view of the results presented concerning the relationship between the LPN and DLPN problems (section 3.1), it is clear that the security of the DLPN construction fundamentally relies on the non-linear properties of g_τ.

In this case the most viable cryptanalytic approach is to attempt an *algebraic attack*. This may be possible if the degree and/or complexity of the non-linear component is not sufficiently high. And indeed this was the attack method used against the DLPN versions proposed in [2]. We also refer to Section 4.1 of [23] for a discussion on the resistance of DLPN constructions against algebraic attacks.

Regarding the instances introduced in this paper, we confidently assert that an algebraic attack is infeasible against two of our instantiations, namely when using ASCON and Simpira (even if followed by no folding). This follows straightforwardly from the security assuarnces provided by the two ciphers. For the case

of LightMAC, we are not able to infer this directly based on the provable properties of the MAC construction – since we do not use a PRP, but rather the **AES4**. However, even after a single folding, the polynomial expression of the noise function will be highly complex and of large degree – as far as we are aware, **AES4** is not susceptible to algebraic attacks. Therefore we feel again confident to claim that our LightMAC-based construction is also resistant to algebraic attacks. These arguments are of course heuristic, but explicitly describing the output bits of cryptographic algorithms such as **AES4** or ASCON is not feasible in practice. In comparison, the security assessment of the construction based on the secret matrix operation followed by UTribes/SIPSER requires a careful analysis of specific parameter choices (as illustrated by the attack in [2]). Based on this discussion, we are confident that our DLPN constructions are resistant to algebraic attacks.

DLPN instantiations must also to be assessed against the standard attacks against LPN, for example based on the BKW algorithm. In turn, such an analysis leads to the selection of suitable parameters ℓ and τ. In this context, we note the assumption made in our constructions that the non-linear component g is a τ-unbalanced weak pseudorandom function. This will be the case for the Folder_τ functions defined in Section 4.1 under the assumption that their input – the output bits of the function Diffuse – are independent and uniformly random. Again, we are confident in claiming that our instantiations with ASCON, Simpira and LightMAC[**AES4**] satisfy these assumptions.

In conclusion, we are highly confident that our DLPN construction and instantiations are secure against algebraic and BKW-based attacks. In our analysis, we did not identify any additional attack avenues against our construction in the weak PRF setting.

6 Experimental results

In this section we describe the implementation of our deterministic LPN framework using the functions discussed in Section 4. We implement $g_\tau = \mathsf{F}_\tau \circ \mathsf{D}$, for various values of τ, ℓ where $\mathsf{F}_\tau = \mathsf{UTribes}$, and D is one of the candidate functions from Section 4.2. The source code is available at this link.

6.1 Platform choices

The different options for Diffuse are chosen so that they are expected to perform well in software. However, performance is of course platform-dependent and affected by many factors. For instance Ascon-PRF has optimised implementations for different platforms, while performance of AES-round based Diffuse function is directly dependent on availability of AES-round hardware support.

Most modern processors have hardware accelleration for AES round, most notably AES-NI on processors with x86 architechture. But many IoT and embedded devices (where LPN based protocols are highly relevant) use processors with ARM-based architecture. Hence we are interested in learning the performance of

AES on such processors. AES hardware support is available for some ARM-based processors, as part of *Cryptographic Extensions* instruction set. However, it is not present on the chip by default and is up to manufacturers whether to include the license for this support. Most mobile phones with ARM processors include this but SoC devices, like Raspberry Pi 3, 4, don't [36]. When explicit AES hardware support is lacking, the polynomial instruction may still be used for efficient AES implementations if ARM NEON is available [41]. Finally, although matrix multiplication is a simple linear operation in theory, its implementation – particularly for large dimensions – can be cumbersome. To the best of our knowledge, no optimised implementation exists that is both portable across devices and competitive with complex cryptographic functions at large scales.

With this in mind, we ran our experiments on two devices to obtain diverse benchmarks. On a AMD EPYC server with 24-core processor and `x86_64` architecture, a server-grade processor with AES-NI that demonstrates the benefit of hardware AES acceleration. The second platform is a Raspberry Pi 3B with ARM Cortex-A53 processor, running in 32 bit mode. This device was chosen to test our results on a setup resembling IoT devices. While the device does not have any hardware acceleration, we note the availability of AES optimisation across processors. Moreover, using a pure C implementation on a platform without AES support actually showcases the huge effect of AES hardware support.

6.2 Implementation details

Overall benchmarks. To measure the CPU cycle count, we use `__rdtscp()` on the x86 platform and `perf_event_open()` on the Raspberry Pi. For each candidate D, we keep the vector length ℓ and the LPN secret s constant. By one evaluation of DLPN we mean that we sample a binary vector a of length ℓ uniformly at random and compute the $g_\tau(s, a) = \mathsf{D} \circ \mathsf{UTribes}$, for different choices of D. We run a quarter of evaluations of DLPN in the cache warm-up phase. Then start the CPU cycle counter and run the evaluations for all samples and calculate the average number of CPU cycles, This experiment is repeated five times and the lowest average CPU cycle count is reported. We also measure the total experiment time (in ms) using `clock_gettime` and memory (in MB) using `getrusage`. More details are provided in the linked GitHub repository. We repeat the experiment for $\ell \in \{512, 1024, 2048, 4096\}$ and $\tau \in \{0.01, 0.05, 0.1, 0.33\}$. We report partial results in Table 1, and in full in Appendix E. The exact description of $\mathsf{UTribes}$ for the selected values of τ can be found in Section 4.

Implementation choices. For the AMD server, we use the reference implementation of AsconPRF[8]. We wrote our own implementation for LightMAC using AES-NI intrinsics in C, whereas the reference implementation of Simpira[9] also utilises these. For Raspberry Pi, due to lack of AES support, we use TinyAES to write our own implementations for Simpira and LightMAC. For Ascon on the other hand, we use the relevant optimised implementation. For the

[8] https://github.com/ascon/ascon-c/tree/main/crypto_auth/asconprfv13/ref
[9] https://mouha.be/simpira/

$\ell = 1024$	$\tau = 0.33$ (x86)			$\tau = 0.05$ (x86)			$\tau = 0.33$ (RPi3)			$\tau = 0.05$ (RPi3)		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	607	26.4	13	644	28	13	42559	353	2	42742	372	2
Ascon	1439	62.3	13	1480	64.5	13	6854	87	2	7037	84	2
LightMAC[**AES4**]	713	31	101	757	32.9	101	38016	282	2	38199	347	2
Matrix	7574	329	161	7793	339	161	488474	3668	2	488655	3497	2
$\ell = 2048$	$\tau = 0.33$ (x86)			$\tau = 0.05$ (x86)			$\tau = 0.33$ (RPi3)			$\tau = 0.05$ (RPi3)		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	1132	49	26	1171	51	26	87695	694	3	87878	696	3
Ascon	2220	96.7	26	2275	99.1	26	10782	108	3	10965	116	3
LightMAC[**AES4**]	1253	54.6	201	1295	56.4	201	71367	416	3	71550	513	3
Matrix	15253	664	230	15671	682	230	196305	14096	4	196324	14103	4

Table 1. Experimental results for x86-64 and RPi3

case where Diffuse is a linear map, i.e. multiplication by a fixed $\ell \times \ell$ matrix, we use the M4RI library for matrix multiplication, which appears to be one of the most well-known libraries supporting optimised matrix operations.[10]. However, we could not find a similar optimised library for binary matrix-vector multiplication on Raspberry Pi. Hence, we used our own C implementation, with ARM NEON intrinsic in C for bitwise XOR. Exact specification of how we instantiate each Diffuse is given in Appendix C.

6.3 Results and Discussion

Comparison of Diffuse instantiations. Across the benchmarks in Table 1, Table 2, and Appendix E, we observe that implementing D with Simpira and LightMAC[**AES4**] incurs the least computation costs for producing deterministic LPN instances. This is clearly due to their use of AES-NI. Simpira is slightly more efficient, which is somewhat surprising given that we instantiate LightMAC with AES4. However, as Simpira was designed for processing large block lengths in mind, this potentially explains the high performance. Furthermore, Simpira's reference implementation may benefit from optimisations that are lacking in our LightMAC implementation. The performances drop drastically with Raspberry Pi 3, which demonstrated the benefit of having AES support. A factor may also be that we wrote our own implementation using TinyAES, and several optimisations could still be possible. Finally, we also note the availability of optimised AES implementations (both with and without AES hardware support) as discussed in Section 6.1 and optimised Simpira for low-end devices [65].

Ascon-PRF remains competitive across both platforms, and as a NIST standard, it offers stronger security guarantees. Ascon is a solid, conservative choice

[10] https://github.com/malb/m4ri/

	$\ell = 512, \tau = 0.077$			$\ell = 1024, \tau = 0.019$			$\ell = 2048, \tau = 0.001$		
	Cycles	Time (ms)	Mem. (MB)	Cycles	Time (ms)	Mem. (MB)	Cycles	Time (ms)	Mem. (MB)
Matrix + SIPSER	5848	254	119	11107	483	141	20896	910	185

Table 2. Experimental results for [23] WPRF

for instantiating DLPN. Furthermore, using the platform-specific optimised implementations of Ascon-PRF (instead of the reference implementation), may lead to improved performances and portability on large range of platforms.

Considering the matrix-based low-depth WPRF constructions, our implementation of the UTribes variant is much less efficient than all other choices. Our M4RI-based implementation incurs an order of magnitude performance slow-down in CPU cycles, when compared with the AES-NI-based constructions. From a memory perspective, while Simpira, LightMAC[**AES4**], and Ascon-PRF are essentially equivalent, the memory costs of the WPRF constructions appear to be around an order of magnitude higher. Overall, it is possible that our M4RI-based implementation is not optimal; the same can be said about our ARM NEON implementation. However, it is unlikely that any optimisation would reduce the gap in performance for matrix multiplication. Moreover, the lack of optimised implementation of linear maps for different platforms demonstrates the difficulty of using D as matrix, when we are interested in instantiating DLPN in practice. Overall, we believe these numbers give an accurate representation of the overall performance and how it compares to other Diffuse.

Note that, for a fixed ℓ, changing the value of τ affects the measurements only slightly for all choices of Diffuse. The memory usage remains the same, while CPU cycles and time incur only slight variations. We believe this is because UTribes is instantiated with different tribe-lengths and number of tribes for different τ. For example, the number of tribes and total tribe length for $\tau = 0.1$ is $4, 26$ and for $\tau = 0.05$ is $5, 41$, resp. Thus UTribes $_{0.05}$ is slightly slower than UTribes $_{0.1}$. Optimisations could potentially be introduced to improve the performance of UTribes further. However, since we implement Diffuse and UTribes separately and sequentially, the gain will be equal across all Diffuse instantiations.

WPRF comparison. For completeness, given that our UTribes formulation of g_τ for processing linear maps outputs may be susceptible to cryptanalysis, we also produced an implementation of SIPSER-based WPRF from [23]. Results are shown in Table 2. As noted previously, our UTribes is indeed a lower bound for performance, and the SIPSER-based construction is indeed much slower. Furthermore, the configurability of τ is less fine-grained, and it is harder to achieve higher values. Our results indicate that instantiating WPRFs from τ-UWPRFs, using the transformation detailed in Lemma 1, is better served using the function compositions built from symmetric primitives (as opposed to current candidate low-depth WPRFs).

7 Acknowledgments

This work was supported by FCT and the CMU Portugal program through the project "Fully-Homomorphic Encryption from Post-Quantum Code-Based Assumptions", ref. 2024.12595.CMU (DOI: 10.54499/2024.12595.cmu), and the LASIGE Research Unit, ref. UID/00408/2025 - LASIGE. Initial drafts of this work were also supported by NOVA.ID.FCT through the CertiCoLab project, funded by the EEA Grants Bilateral Fund (FBR_OC2_103-CertiCoLab).

References

1. M. Ajtai and N. Linial. The influence of large coalitions. *Comb.*, 13(2):129–145, 1993.
2. A. Akavia, A. Bogdanov, S. Guo, A. Kamath, and A. Rosen. Candidate weak pseudorandom functions in ac0 . mod2. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, page 251260, New York, NY, USA, 2014. Association for Computing Machinery.
3. J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs. Learning with rounding, revisited - new reduction, properties and applications. In R. Canetti and J. 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 57–74. Springer, 2013.
4. B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. Secure arithmetic computation with constant computational overhead. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, 2017.
5. F. Armknecht, M. Hamann, and V. Mikhalev. Lightweight Authentication Protocols on Ultra-Constrained RFIDs - Myths and Facts. In N. Saxena and A.-R. Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues*, pages 1–18, Cham, 2014. Springer International Publishing.
6. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In D. Pointcheval and T. 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.
7. S. Banik, T. Isobe, F. Liu, K. Minematsu, and K. Sakamoto. Orthros: A low-latency prf. *IACR Transactions on Symmetric Cryptology*, 2021(1):3777, Mar. 2021.
8. A. Bariant, J. Baudrin, G. Leurent, C. Pernot, L. Perrin, and T. Peyrin. Fast AES-based universal hash functions and macs: Featuring lemac and petitmac. *IACR Transactions on Symmetric Cryptology*, 2024(2):3567, Jun. 2024.
9. C. Baum, L. Braun, C. D. de Saint Guilhem, M. Klooß, C. Majenz, S. Mukherjee, E. Orsini, S. Ramacher, C. Rechberger, L. Roy, et al. FAEST: algorithm specifications. Technical report, National Institute of Standards and Technology, 2023.
10. C. Baum, S. Dittmer, P. Scholl, and X. Wang. SOK: vector OLE-based zero-knowledge protocols. *Des. Codes Cryptogr.*, 91(11):3527–3561, 2023.
11. D. Bellizia, C. Hoffmann, D. Kamel, H. Liu, P. Méaux, F. Standaert, and Y. Yu. Learning parity with physical noise: Imperfections, reductions and FPGA prototype. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):390–417, 2021.

12. D. Bellizia, C. Hoffmann, D. Kamel, P. Méaux, and F. Standaert. When bad news become good news towards usable instances of learning with physical errors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):1–24, 2022.
13. A. Ben-Efraim, K. Cong, E. Omri, E. Orsini, N. P. Smart, and E. Soria-Vazquez. Large scale, actively secure computation from LPN and free-xor garbled circuits. In A. Canteaut and F. 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 III*, volume 12698 of *LNCS*, pages 33–63. Springer, 2021.
14. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.
15. A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.
16. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In F. F. Yao and E. M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 435–440. ACM, 2000.
17. A. Bogdanov, M. M. Lauridsen, and E. Tischhauser. AES-based authenticated encryption modes in parallel high-performance software. *IACR Cryptol. ePrint Arch.*, page 186, 2014.
18. A. Bogdanov and A. Rosen. Pseudorandom functions: Three decades later. Cryptology ePrint Archive, Paper 2017/652, 2017. https://eprint.iacr.org/2017/652.
19. D. Boneh, Y. Ishai, A. Passelègue, A. Sahai, and D. J. Wu. Exploring crypto dark matter: - new simple PRF candidates and their applications. In A. Beimel and S. Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 699–729. Springer, 2018.
20. D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. In R. Canetti and J. 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.
21. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 896–912. ACM, 2018.
22. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In S. Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1069–1080. IEEE, 2020.
23. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Low-complexity weak pseudorandom functions in AC0[MOD2]. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, 2021, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 487–516. Springer, 2021.

24. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Low-complexity weak pseudorandom functions in AC0[MOD2], 2021. Full version, made available privately by the authors.
25. C. Carlet and P. Sarkar. Use of simple arithmetic operations to construct efficiently implementable boolean functions possessing high nonlinearity and good resistance to algebraic attacks. Cryptology ePrint Archive, Paper 2024/1305, 2024.
26. J. H. Cheon, W. Cho, J. H. Kim, and J. Kim. Adventures in crypto dark matter: Attacks and fixes for weak pseudorandom functions. In J. A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, May 10-13, 2021, Proceedings, Part II*, volume 12711 of *LNCS*, pages 739–760. Springer, 2021.
27. G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference 2021, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534. Springer, 2021.
28. Q. Dao, Y. Ishai, A. Jain, and H. Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 315–348. Springer, 2023.
29. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.
30. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon MAC, PRF, and Short-Input PRF - Lightweight, Fast, and Efficient Pseudorandom Functions, 2024.
31. C. Dobraunig, B. Mennink, and S. Neves. Elimac: Speeding up lightmac by around 20%. *IACR Trans. Symmetric Cryptol.*, 2023(2):69–93, 2023.
32. N. Döttling, J. Müller-Quade, and A. C. A. Nascimento. IND-CCA secure cryptography based on a variant of the LPN problem. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 485–503. Springer, 2012.
33. N. Drucker, S. Gueron, and V. Krasnov. Making AES great again: the forthcoming vectorized AES instruction. Cryptology ePrint Archive, Paper 2018/392, 2018.
34. P. A. Eliasi, Y. Belkheyar, J. Daemen, S. Ghosh, D. Kuijsters, A. Mehrdad, S. Mella, S. Rasoolzadeh, and G. V. Assche. Koala: A low-latency pseudorandom function. Cryptology ePrint Archive, Paper 2024/1249, 2024.
35. A. Esser, R. Kübler, and A. May. LPN decoded. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 486–514. Springer, 2017.
36. H. Fujii, F. C. Rodrigues, and J. López. Fast AES implementation using armv8 asimd without cryptography extension. In J. H. Seo, editor, *Information Security and Cryptology – ICISC 2019*, pages 84–101, Cham, 2020. Springer.
37. H. Gilbert, M. Robshaw, and H. Sibert. Active attack against HB+: a provably secure lightweight authentication protocol. *Electronics Letters*, 41:1169–1170, 2005.

38. H. Gilbert, M. J. B. Robshaw, and Y. Seurin. HB#: increasing the security and efficiency of HB+. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, page 361378, Berlin, Heidelberg, 2008. Springer-Verlag.
39. H. Gilbert, M. J. B. Robshaw, and Y. Seurin. How to Encrypt with the LPN Problem. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming*, pages 679–690, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
40. S. Gilboa and S. Gueron. The advantage of truncated permutations. *Discrete Applied Mathematics*, 294:214223, May 2021.
41. C. P. L. Gouvêa and J. López. Implementing GCM on ARMv8. In K. Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 167–180, Cham, 2015. Springer.
42. R. Gross. Generalized tribes. Boolean Zoo, Weizmann Institute, 2019.
43. S. Gueron and N. Mouha. Simpira v2: A family of efficient permutations using the AES round function. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 95–125, 2016.
44. G. Hammouri and B. Sunar. PUF-HB: A Tamper-Resilient HB Based Authentication Protocol. In S. M. Bellovin, R. Gennaro, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security*, pages 346–365, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
45. J. Håstad. Almost optimal lower bounds for small depth circuits. In J. Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986.
46. J. Håstad. Some optimal inapproximability results. In F. T. Leighton and P. W. Shor, editors, *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 1–10. ACM, 1997.
47. L. Heimberger, D. Kales, R. Lolato, O. Mir, S. Ramacher, and C. Rechberger. Leap: A fast, lattice-based OPRF with application to private set intersection. In S. Fehr and P. Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VII*, volume 15607 of *Lecture Notes in Computer Science*, pages 254–283. Springer, 2025.
48. S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. Lapin: An efficient authentication protocol based on ring-lpn. In A. Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2012.
49. V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.
50. N. J. Hopper and M. Blum. Secure human identification protocols. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.

51. J. Jean and I. Nikolic. Efficient design strategies based on the AES round function. In T. Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.
52. A. Juels and S. A. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
53. D. Kamel, D. Bellizia, O. Bronchain, and F. Standaert. Side-channel analysis of a learning parity with physical noise processor. *J. Cryptogr. Eng.*, 11(2):171–179, 2021.
54. D. Kamel, D. Bellizia, F.-X. Standaert, D. Flandre, and D. Bol. Demonstrating an LPPN Processor. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, ASHES '18, page 1823, New York, NY, USA, 2018. ACM.
55. D. Kamel, F. Standaert, A. Duc, D. Flandre, and F. Berti. Learning with physical noise or errors. *IEEE Trans. Dependable Secur. Comput.*, 17(5):957–971, 2020.
56. C. Lefevre, Y. Belkheyar, and J. Daemen. Kirby: A robust permutation-based PRF construction. Cryptology ePrint Archive, Paper 2023/1520, 2023.
57. A. Luykx, B. Preneel, E. Tischhauser, and K. Yasuda. A MAC mode for lightweight block ciphers. In T. Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *LNCS*, pages 43–59. Springer, 2016.
58. M. Madhavan, A. Thangaraj, Y. Sankarasubramanian, and K. Viswanathan. NLHB: A non-linear Hopper-Blum protocol. In *2010 IEEE International Symposium on Information Theory*, pages 2498–2502, 2010.
59. B. Mennink and S. Neves. Optimal PRFs from Blockcipher Designs. *IACR Transactions on Symmetric Cryptology*, 2017(3):228252, Sep. 2017.
60. K. Pietrzak. Cryptography from learning parity with noise. In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, editors, *SOFSEM 2012: 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012.
61. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.
62. B. Rossman, R. A. Servedio, and L. Tan. An average-case depth hierarchy theorem for boolean circuits. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1030–1048. IEEE Computer Society, 2015.
63. J. K. Rott. Intel advanced encryption standard instructions (AES-NI). Blog, 2012.
64. L. Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Y. Dodis and T. Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 657–687. Springer, 2022.
65. M. Sim, S. Eum, H. Kwon, K. Jang, H. Kim, H. Kim, G. Song, W. Lee, and H. Seo. Optimized implementation of simpira on microcontrollers for secure massive learning. *Symmetry*, 14(11), 2022.

66. M. Sönmez Turan, K. McKay, D. Chang, J. Kang, and J. Kelsey. Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions. Technical report, National Institute of Standards and Technology, 2024. NIST SP 800-232.

A Overview of HB-family protocols

The HB family of protocols, which include a series of work starting with Hopper and Blum [50] are authentication protocols specifically designed for resource-constrained devices such as RFID tags. These protocols typically contain a challenge-response sequence that resembles an LPN instance, thus allowing them to leverage the simplicity of LPN and security of the LPN problem. In these protocols, a prover device (referred to as *tag*) proves the knowledge of a shared key $s \in \mathbb{Z}_2^n$ to the verifier device (referred to as reader *reader*). In the first and the simplest protocol (HB) [50], the reader sends challenges $a_i \in \mathbb{Z}_2^n$ and the tag responds with the bit $r_i = \langle a_i, s\rangle + e_i$, where $e_i \leftarrow \mathsf{Ber}_\tau$. This is repeated for m samples $a_1, \ldots, a_m$ and the verifier, using the shared secret s, then checks if $r_i = \langle a_i, s\rangle$. Taking into account the noise rate τ, the tag has authenticated successfully if the check fails at most $\lfloor \tau m \rfloor$ many times. It is clear that this challenge-response resembles an LPN instance, leading to security against *passive* adversaries. However, an *active* adversary can repeatedly query the same challenge vector v, and naively guess the value $\langle s, v\rangle$ by majority rule, as the error is only added with probability $\tau < 1/2$. Repeating this for n linearly independent vectors $v_1, \ldots, v_n$ leads to easily recovering the n-bit secret s.

The follow up work of HB+ [52] protocol set up a similar protocol as HB but with two shared secret bit-strings, instead of one. The work claimed security against active adversaries, again by reduction to the hardness of LPN, but the proof was shown to be flawed [37] through a Man-in-the-Middle (MITM) attack. There have been several other variants of similar authentication protocols. We refer to [5, Appendix A] for a summary.

B Folding Function Example Parametrisations

Given the $\mathsf{UTribes}$ function described in Section 4.1, we provide examples of instantiations of F_τ in DNF formula, that allow us to realise the function g_τ, for various values of τ. Note that the function formulations are derived explicitly from the construction in Section 4.1, and vary with the value of τ sought.

$\tau = 0.33$**:** Number of tribes = 5, and tribe sizes $\omega_0 = 2, \omega_1 = 4, \omega_2 = 5, \omega_3 = 6, \omega_4 = 11$.

$$g_{\frac{1}{3}}(x_1, \ldots, x_n) = \left(\bigwedge_{i=1}^{2} x_i\right) \vee \left(\bigwedge_{i=3}^{6} x_i\right) \vee \left(\bigwedge_{i=7}^{11} x_i\right) \vee \left(\bigwedge_{i=12}^{17} x_i\right) \vee \left(\bigwedge_{i=18}^{28} x_i\right)$$

Given that the next tribe length would be $\omega_6 = 12$, this gives an approximation error of $\epsilon_5 < 2^{-11}$ (Equation (4)). Total number of bits used = 28.

$\tau = 0.1$**:** Number of tribes $= 4$, and tribe sizes $\omega_0 = 4, \omega_1 = 5, \omega_2 = 7, \omega_3 = 10$.

$$g_{\frac{1}{10}}(x_1, \ldots, x_n) = \left(\bigwedge_{i=1}^{4} x_i\right) \vee \left(\bigwedge_{i=5}^{9} x_i\right) \vee \left(\bigwedge_{i=10}^{16} x_i\right) \vee \left(\bigwedge_{i=17}^{26} x_i\right)$$

Given that the next tribe length would be $\omega_5 = 12$, this gives an approximation error of $\epsilon_4 < 2^{-11}$. Total number of bits used $= 26$.

$\tau = 0.05$**:** Number of tribes $= 5$, with sizes $\omega_0 = 5, \omega_1 = 6, \omega_2 = 9, \omega_3 = 10, \omega_4 = 11$

$$g_{\frac{1}{20}}(x_1, \ldots, x_n) = \left(\bigwedge_{i=1}^{5} x_i\right) \vee \left(\bigwedge_{i=6}^{11} x_i\right) \vee \left(\bigwedge_{i=12}^{20} x_i\right) \vee \left(\bigwedge_{i=21}^{30} x_i\right) \bigvee \left(\bigwedge_{i=31}^{41} x_i\right)$$

Given that the next tribe length would be $\omega_6 = 12$, this gives an approximation error of $\epsilon_5 < 2^{-11}$. Total number of bits used $= 41$.

$\tau = 0.01$**:** Number of tribes $= 4$, with sizes $\omega_0 = 7, \omega_1 = 9, \omega_2 = 12, \omega_3 = 17$

$$g_{\frac{1}{100}}(x_1, \ldots, x_n) = \left(\bigwedge_{i=1}^{7} x_i\right) \vee \left(\bigwedge_{i=8}^{16} x_i\right) \vee \left(\bigwedge_{i=17}^{28} x_i\right) \vee \left(\bigwedge_{i=29}^{45} x_i\right)$$

Given that the next tribe length would be $\omega_6 = 22$, this gives an approximation error of $\epsilon_4 < 2^{-21}$. Total number of bits used $= 41$.

C Implementation specifications for Diffuse

Simpira. We generate a vector $s \in \{0,1\}^\ell$ which is cast as an `_m128i` variable. This is the LPN secret and is fixed. For each iteration, we generate LPN sample $a_i \in \{0,1\}^\ell$ (again as `_m128i` variable) and is mix with the LPN secret by bitwise XOR of both, and then apply the Simpira permutation as Diffuse. For this, use the reference implementation of Simpira [11]. The file `Permutations_Ref.c` contains reference implementations for all values of B (here B is the number of 128-bit word in the input vector), which utilize the `aesenc` native instruction (hence the need for generating inputs as `_m128i`). The n-bit output vector (which is in `_m128i` format) is the unpacked and we apply UTribes to get the noise bit. Specifically, for each LPN sample of length n we apply the corresponding Simpira permutation S_B where $B = \ell/128$ then compute the noise term as

$$e_i = \mathsf{UTribes}(S_B(s \oplus a_i))$$

Ascon-PRF. We use the reference implementation of Ascon-PRF, but do not use any platform-specific optimisations.[12] We generate a vector $s \in \{0,1\}^\ell$ as a byte array. This is the LPN secret and is fixed. For each iteration, we generate LPN sample $a_i \in \{0,1\}^\ell$ as a byte array. Then we apply the Ascon-PRF with

[11] https://mouha.be/simpira/

[12] https://github.com/ascon/ascon-c/tree/main/crypto_auth/asconprfv13/ref

input a_i, key s, input-length and key-length (in bytes) is ℓ in bytes and output length is 16 bytes. Then we convert the output in an array of bits and apply UTribes to the first few bits (≤ 64) to get the noise bit. Specifically, the noise bit e_i is given as:

$$e_i = \mathsf{UTribes}(F(s, a_i, \text{inlen} = \ell, \text{outlen} = 128)).$$

We note that we experiment also with outlen = 256, and show the results in the first table shown. However the results indicate that the performance degrades expectably, and so we do not repeat the results for other dimensions.

LightMAC. We use the LightMAC construction as described in Section 4.2 (i.e. with a similar design philosophy to that of EliMAC [31]), but we note that UTribes only uses an initial sequence of bits (≤ 64) of the final output, thus we do not need to specifically perform the truncation that occurs in the final round. Furthermore, for the block cipher encryption function, E, we use **AES4** as described in [49]. That is, for sub-keys $K = (K_0, K_1, K_2, K_3, K_4)$ and input $X \in \{0,1\}^{128}$

$$\mathbf{AES4}(K, X) = \texttt{aesenc}(\texttt{aesenc}(\texttt{aesenc}(\texttt{aesenc}(X \oplus K_0, K_1), K_2), K_3), K_4)$$

where `aesenc` is one AES round (where we use the native instruction). We generate a 128-bit master key s (this can be derived from the LPN secret), and use the AES-key expansion to get 10 sub-keys $(K_0, \ldots, K_9)$, and use the first 4 sub-keys during E_{k_1} and last 5 sub-keys during E_{k_2} in LightMAC

We make two choices for cleaner implementations. In the original construction, the user has a choice of how this s-bit vector i_s is determined based on i. For instance, it could be the s bit representation of i (Section 2, [57]). Additionally, s should be chosen such that the input message M is of length at most $2^s(n_1 - s)$ bits. We choose $s = 8$ and i_8 to be the 8-bit representation of i. This choice is made for simplicity's sake, so that, for every index i, we can then cast the integer value of i as an unsigned byte and concatenate with 15 bytes taken from the message stream and encrypt the 16 bytes with **AES4**. Since we choose $E = \mathbf{AES4}$, then $n_1 = 128$ and the maximum message length is $2^8 \times 120$, which is suitable for our choices of LPN dimensions.

Matrix-based Diffuse with UTribes. For the case where Diffuse is a linear map i.e. multiplication by a fixed $\ell \times \ell$ matrix. We use the M4RI library for fast matrix multiplication, which appears to be one of the most well-known libraries supporting optimised matrix multiplication operations.[13] We generate an $\ell \times \ell$ matrix M (in the `mzd_t` datatype native to the M4RI library), and keep it fixed. For each iteration, we generate the LPN vectors $\boldsymbol{a}_i \in \{0,1\}^\ell$ by casting the random bytes in an array and pointing an `mzd_t` vector `v` to it. Then we multiply M with `v` using `_mzd_mul_va` function from the library. For Raspberry Pi, we could not find such a library for optimized linear maps. We wrote our own implementation where we utilized ARM NEON intrinsics for performing bitwise XOR between each matrix row and $\boldsymbol{a}_i$, and to get the parity of the number of 1's

[13] https://github.com/malb/m4ri/

in the resulting vector, we use `libpopcount.h`, a portable library for population count. We read the bits from the resulting output `mzd_t`-type vector, and apply UTribes to the first few bits (≤ 64, see example parametrisations in Section 4).

Matrix-based Diffuse with SIPSER. While the case above should lower bound the performance of the WPRF from [23], we test a full implementation that uses the SIPSER function applied after linear map for x86 platform. Note that the SIPSER formulation is the one that would appear to maintain security against known cryptanalysis. This variant is determined by a parameter λ which relates to the input length ℓ, and as discussed previously, the error rate τ is a function of λ. For LPN vector length $\ell \in \{512, 1024, 2048\}$, the closest we get is $\lambda \in \{16, 22, 32\}$ respectively, and this implies that the approximate τ that is used in our experiments is $\{0.077, 0.019, 0.001\}$ respectively.

D Applications of DLPN

Relationship with VOLE. LPN has become a vital building block in the construction of primitives based on vector oblivious linear evaluation (VOLE) [22]. VOLE itself lends itself to practical instantiations of signatures schemes [9], zero-knowledge proofs [10], oblivious pseudorandom functions [47], and oblivious transfer [64], amongst many other use-cases. With VOLE seeing increased practical usage, this requires sampling secure instances of LPN for building similarly secure instances of VOLE. One of the appealing aspects of VOLE-based protocols is that their computational load is fairly low, and this leads to zero-knowledge proofs with lower prover overheads (as opposed to zkSNARKs [14], for example). This motivates incorporating such applications with lower-powered clients. As a result, we see a potentially important application of our DLPN framework in instantiating VOLE-based applications, where constrained clients may be asked to sample many LPN instances that each require valuable sources of randomness. This would allow clients to build VOLE instances, with security dependent on the parameterisation and construction of DLPN that is used.

Large-scale MPC. The work of [13] demonstrates how to build generic multi-party computation from LPN, noting that specific error rates (e.g. 1/8 or 1/16) are required to ensure that the computation remains correct. Such secure computation frameworks appear to be efficient at large-scale, and also ensure active security, which is preferable for higher-risk applications. DLPN here would allow specifically configuring error rates to the required values, while permitting that such computation systems can be built without sampling sources of randomness. Such an instantiation could be useful for building federated learning platforms, requiring computation over private data by large numbers of constrained clients.

E Full experimental results

$\ell = 512$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	430	18.7	7	437	19	7	465	20.2	7	472	20.5	7
Ascon-128	1018	44.3	7	1025	44.7	7	1053	45.8	7	1078	46.9	7
Ascon-256	1196	52.1	7	1200	52.3	7	1237	53.8	7	1240	54.9	7
LightMAC[**AES4**]	477	20.8	7	470	22.4	7	513	22.3	7	517	22.5	7
Matrix	5104	222	128	5002	217	128	5011	218	128	5049	219	128

$\ell = 1024$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	607	26.4	13	627	27	13	644	28	13	656	28.5	13
Ascon	1439	62.3	13	1433	62.4	12	1480	64.5	13	1488	64.8	13
LightMAC[**AES4**]	713	31	101	724	31	101	757	32.9	101	762	33.2	101
Matrix	7574	329	161	7569	329	161	7793	339	161	7566	329	161

$\ell = 2048$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	1132	49	26	1148	50	26	1171	51	26	1179	51	26
Ascon	2220	96.7	26	229	97	26	2275	99.1	26	2285	99.5	26
LightMAC[**AES4**]	1253	54.6	201	1265	55.1	201	1295	56.4	201	1305	56.8	201
Matrix	15253	664	230	15174	661	230	15671	682	230	15043	655	230

$\ell = 4096$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	2057	89.6	51	2063	90	51	2098	91	51	2116	92	51
Ascon	3855	167	51	3858	168	51	3914	170	51	3916	170	51
LightMAC[**AES4**]	2287	99.3	51	2265	98.6	51	2328	101	51	2323	101	51
Matrix	67391	2935	369	66292	2887	369	67515	2940	369	67683	2948	369

Table 3. Experimental results for x86-64.

$\ell = 512$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	17674	192	1	17674	174	1	17857	173	1	17913	172	1
Ascon-128	4889	72	1	4862	73	1	5072	75	1	5128	88	1
Ascon-256	5804	80	1	5777	82	1	5987	74	1	6043	96	1
LightMAC[**AES4**]	23182	209	1	23155	219	1	23364	192	1	23421	207	1
Matrix-ℓ	125188	994	1	125160	959	1	125369	936	1	125427	940	1
Matrix-128	32039	294	1	32011	295	1	32222	282	1	32277	296	1

$\ell = 1024$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	42559	353	2	42532	355	2	42742	372	2	42798	379	2
Ascon-128	6854	87	2	6827	97	2	7037	84	2	7093	98	2
Ascon-256	7769	82	2	7742	87	2	7952	90	2	8008	94	2
LightMAC[**AES4**]	38016	282	2	37989	268	2	38199	347	2	38255	265	2
Matrix-ℓ	488474	3668	2	488445	3692	2	488655	3497	2	488717	3533	2
Matrix-128	61934	509	2	61907	503	2	62116	480	2	62173	5066	2

$\ell = 2048$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	87695	694	3	87669	700	3	87878	696	3	87934	688	3
Ascon-128	10782	108	3	10755	117	3	10965	116	3	11021	101	3
Ascon-256	11697	94	3	11670	113	3	11880	112	3	11936	114	3
LightMAC[**AES4**]	71367	416	3	71340	490	3	71550	513	3	71606	479	3
Matrix-ℓ	1963057	14096	4	1963030	14102	4	1963241	14103	4	1963329	14232	4
Matrix-128	123647	927	3	123619	974	3	123828	947	3	123885	950	3

$\ell = 4096$	$\tau = 0.33$			$\tau = 0.1$			$\tau = 0.05$			$\tau = 0.01$		
	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)	Cycles	Time (ms)	Memory (MB)
Simpira	183386	1376	6	183359	1388	6	183569	1396	6	183628	1375	6
Ascon-128	18641	128	6	18641	148	6	18824	127	6	18880	130	6
Ascon-256	19556	124	6	19529	132	6	19739	128	6	19795	150	6
LightMAC[**AES4**]	134369	866	6	134344	919	6	134552	933	6	134608	964	6
Matrix-ℓ	7390247	55704	8	7390223	52778	8	7390435	52342	8	7390486	52326	8
Matrix-128	231908	1645	6	231880	1643	6	232091	1607	6	232146	1603	6

Table 4. Experimental results for Raspberry Pi 3.