IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 3, pp. 390–417. DOI:10.46586/tches.v2021.i3.390-417

Learning Parity with Physical Noise: Imperfections, Reductions and FPGA Prototype

Davide Bellizia[1], Clément Hoffmann[1], Dina Kamel[1], Hanlin Liu[2], Pierrick Méaux[1], François-Xavier Standaert[1] and Yu Yu[2]

[1] UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be
[2] Shanghai Jiao Tong University, China

Abstract. Hard learning problems are important building blocks for the design of various cryptographic functionalities such as authentication protocols and post-quantum public key encryption. The standard implementations of such schemes add some controlled errors to simple (e.g., inner product) computations involving a public challenge and a secret key. Hard physical learning problems formalize the potential gains that could be obtained by leveraging inexact computing to directly generate erroneous samples. While they have good potential for improving the performances and physical security of more conventional samplers when implemented in specialized integrated circuits, it remains unknown whether physical defaults that inevitably occur in their instantiation can lead to security losses, nor whether their implementation can be viable on standard platforms such as FPGAs. We contribute to these questions in the context of the Learning Parity with Physical Noise (LPPN) problem by: (1) exhibiting new (output) data dependencies of the error probabilities that LPPN samples may suffer from; (2) formally showing that LPPN instances with such dependencies are as hard as the standard LPN problem; (3) analyzing an FPGA prototype of LPPN processor that satisfies basic security and performance requirements.

Keywords: Learning Parity with Noise · Physical Assumptions · Physical Defaults · Security Reductions · FPGA Implementations · Side-Channel Security

1 Introduction

For more than two decades, learning in the presence of errors has been considered as an interesting source of computationally hard problems [Kea93, Hås97], which have in turn found many applications in the design of provably secure cryptographic schemes [Reg10, Pie12]. Among the possible instances of such hard problems, the Learning Parity with Noise (LPN) problem is probably the simplest one. It has given rise to a wide literature about efficient authentication protocols, starting with the scheme of Hopper and Blum (HB) [HB01] and extended in various directions [JW05, GRS08a, KPC+11, DKPW12, HKL+12]. It can also be used to build one way-functions [BFKL93], secret-key encryption schemes [GRS08b], public-key encryption schemes [DV13], pseudorandom functions [YS16] or hash functions [YZW+19]. Extensions of the LPN problem turn out to be even more versatile. For example, the Learning With Errors (LWE) problem introduced by Regev [Reg05] can additionally be used to build identity-based encryption [GPV08, CHKP10] or fully homomorphic encryption [Gen09, BV11]. Its variants are the basis for many post-quantum cryptographic schemes [BL17] and are in particular the core assumption for CRYSTALS-Dilithium [DKL+18], CRYSTALS-Kyber [BDK+18], which are both finalists of the NIST Post-Quantum (PQ) cryptography competition.[1] Finally, large-modulus LPN has re-

[1] https://csrc.nist.gov/projects/post-quantum-cryptography.

Received: 2020-10-15 Accepted: 2020-12-15 Published: 2021-07-09

cently found advanced applications in secure multi-party computation [BCGI18, BCG+19], arithmetic cryptography [AAB17] and indistinguishability obfuscation [JLS20].

Whenever such primitives become practical to the point of being deployed, their secure and efficient implementation naturally becomes a critical problem to solve. This is for example the case of LNP-based authentication protocols and PQ schemes. In this respect, various results showed that the conceptual simplicity of LPN-based protocols does not directly translate into concrete advantages [BL12, AHM14]. This is in part due to the cost of generating random numbers that those solutions require. Similarly, while it could be expected that the simple algebraic structure of LPN-based protocols could directly lead to implementations secure against side-channel attacks, it turned out the need to protect its (additive) Bernoulli noise implies significant overheads as well [GLS14].

As a result of these limitations, Kamel et al. observed that by leveraging recent advances in inexact computing (a trend that is of independent interest in view of the miniaturization of electronic devices [GR10]), one could implement LPN-based schemes without explicitly generating random numbers [KSD+20]. They formalized the corresponding problem as the Learning with Physical Noise (LPPN) problem, and highlighted its potential advantages in terms of implementation cost (since it removes the need to explicitly generate randomness for the additive errors) and implementation security (since it prevents the trivial attack probing the randomness in a leaking implementation). A first ASIC prototype of LPPN processor was then described in [KBS+18] and its side-channel security has been evaluated in [KBBS20]. Nevertheless, and despite these promising and quite unique features, the understanding of the security and applicability of LPPN remains limited.

On the one hand, LPPN is a physical assumption (in contrast with the mathematical LPN assumption), raising the question whether the errors generated thanks to inexact computing (which may not exactly follow a Bernoulli distribution) are leading to hard problems? For example, the investigations in [KSD+20, KBS+18] showed that the error probability of LPPN samples may depend on the Hamming weight of the corresponding challenges, and provided heuristic evidence that these dependencies can be limited.

On the other hand, the prototype implementations in these previous works were heavily relying on an accurate control of frequency and voltage over-scaling, raising the question whether such ideas are applicable with off-the-shelf devices like FPGAs?

Our contributions to these questions are threefold:

First, we further study the physical imperfections that can affect the security of the LPPN problem. In particular, we show that besides the dependencies of the LPPN error probability on its input challenges, LPPN prototypes can also suffer from output dependencies. That is, the probability of error of LPPN samples can depend on the (correct) value of its output. We also show that certain types of implementations lead to "structured errors" (i.e., the error at cycle i depends on the output at cycle i-1). We then discuss options to mitigate these issues, which turn out to be easy to fix by design for the structured errors, and hard to completely cancel for the output dependencies. Hence, these observations question the equivalence between LPPN and LPN.

Second, and since it is well known that tweaking the LPN problem can make it weaker [AG10], we formalize the exhibited output dependencies as a (new) LPN problem with Output Dependencies (LPN-OD) and we show that solving the LPN-OD problem is as hard as solving an LPN problem with adapted security parameters (depending on the output dependencies). This reduction allows us to gain provable confidence that certain types of physical imperfections do not completely ruin the hardness of LPPN problems. It also opens the way towards formalizing other imperfections with similar reductions.

Third, we describe and analyze a prototype LPPN processor running on a Xilinx FPGA, showing that inexact computing can also be leveraged on such general-purpose platforms. In particular, we describe an error control mechanism based on a fully digital

variable delay line that can reach a level of control similar to what can be achieved with an ASIC prototype. We additionally analyze the input and output dependencies of the error probability that our implementation leads to, and conclude that secure LPPN samples can be generated with it. A side-channel security evaluation is also given in Appendix.

As a side-result, we finally discuss how much masking the LPPN implementation, which is required in order to reach high security levels against side-channel attacks, can also be used to mitigate the error dependencies of LPPN samples.[2] This observation strengthens the relevance of the LPPN assumption, since it implies additional concrete advantages in the practically important context of side-channel secure implementations.

Overall, the LPPN assumption is admittedly a provocative one, and a lot of work remains needed to gain confidence in its security and usability. Besides, its strongest potential use case is for the side-channel secure implementation of PQ cryptographic algorithms and most of them rely on more general LWE problems (for which a physical counterpart is also outlined in [KSD+20]). Yet, we believe understanding the security of LPPN in front of possible physical imperfections (like input or output error dependencies), and its implementation in FPGAs, are necessary first steps in this direction. We hope our results can be used as a seed towards both the theoretical investigation of other hard physical learning problems and the practical investigation of inexact computing in this context. We also believe such a risky research path is justified by the high importance of obtaining secure and efficient PQ cryptographic implementations in the future.

2 Background

2.1 The LPN problem

The LPN problem can be specified as follows [LF06]:

Definition 1 (LPN problem)**.** Let $\langle ., . \rangle$ denote the binary inner product, $\boldsymbol{k}$ be a random n-bit secret, $\epsilon \in [0, \frac{1}{2}]$ be a noise parameter, Ber_ϵ be the Bernoulli distribution with parameter ϵ (if $e \leftarrow \mathsf{Ber}_\epsilon$, then $\Pr[e = 1] = \epsilon$), and $D^\epsilon_{\boldsymbol{k}}$ be the distribution defined as:

$$D^\epsilon_{\boldsymbol{k}} =: \{(\boldsymbol{x}, \langle \boldsymbol{x}, \boldsymbol{k} \rangle \oplus e) : \boldsymbol{x} \leftarrow \{0,1\}^n; e \leftarrow \mathsf{Ber}_\epsilon\}.$$

Let $\mathcal{O}^\epsilon_{\boldsymbol{k}}$ denote an oracle outputting independent samples according to the distribution $D^\epsilon_{\boldsymbol{k}}$. The LPN^n_ϵ problem is said to be (q, t, m, θ)-hard to solve if for any algorithm A,

$$\Pr[\boldsymbol{k} \leftarrow \{0,1\}^n : A^{\mathcal{O}^\epsilon_{\boldsymbol{k}}}(1^n) = \boldsymbol{k}] \leq \theta,$$

and A runs in time $< t$, with memory $< m$ and makes at most q queries to $\mathcal{O}^\epsilon_{\boldsymbol{k}}$.

2.2 The LPPN problem

The LPPN problem, introduced in [KSD+20], is a variant of the LPN problem where a physical function directly computes incorrect inner products. Its specification requires the definition of physical function adapted from [AMS+11] which we recall next:

Definition 2 (Physical function)**.** A physical function $\mathsf{PF}_{d,\alpha}$ is a probabilistic procedure based on a physical device d, which can be stimulated with an input $\boldsymbol{x} \in \{0,1\}^{n_i}$, making d respond with a (probabilistic) output $\boldsymbol{y} \in \{0,1\}^{n_o}$, with α a set of parameters.

In the LPPN case, d will be an implementation of inexact inner product computations (implying $n_i = n$ and $n_o = 1$), which we define next, and α is the set of parameters determining its error distribution (e.g., the clock frequency or supply voltage).

[2] The high-level intuition that masking helps against such dependencies is already given in [KSD+20] but we further detail it and connect it with side-channel security requirements.

Definition 3 (Physical inner product)**.** Let $\boldsymbol{k} \in \{0,1\}^n$ be a random n-bit secret stored in a device $d_{\boldsymbol{k}}$. A physical function $\mathsf{PF}_{d_{\boldsymbol{k}},\alpha}$ is called an ϵ-Physical Inner Product (ϵ-PIP) if, on uniform public input $\boldsymbol{x} \in \{0,1\}^n$, it outputs $\langle \boldsymbol{x}, \boldsymbol{k} \rangle$ with estimated error probability:

$$\hat{\Pr}[\mathsf{PF}_{d_{\boldsymbol{k}},\alpha}(\boldsymbol{x}) \neq \langle \boldsymbol{x}, \boldsymbol{k} \rangle] = \epsilon.$$

The LPPN problem can then be defined as follows:

Definition 4 (LPPN problem)**.** Let $\mathsf{PF}_{d_{\boldsymbol{k}},\alpha}$ be an ϵ-PIP. The $\mathrm{LPPN}^{n,\epsilon}_{d_{\boldsymbol{k}},\alpha}$ problem is said to be (q,t,m,θ)-hard to solve if for any algorithm A running in time $< t$, memory $< m$ and making at most q uniformly random queries to an ϵ-PIP, it holds that:

$$\Pr[\boldsymbol{k} \leftarrow \{0,1\}^n : A^{\mathsf{PF}_{d_{\boldsymbol{k}},\alpha}}(1^n) = \boldsymbol{k}] \leq \theta.$$

The main difference between the LPN and LPPN problems is that assuming the LPN problem to be hard is a mathematical assumption, while assuming the LPPN problem to be hard is a physical assumption. In particular, we have no guarantee that the LPPN error distribution is exactly equal to the one specified in Definition 1. In the next section, we show that concrete instances of physical functions can be affected by various defaults of the noise distribution, hence questioning the equivalence of these problems.

3 LPPN physical imperfections

It has already been put forward in [KSD+20, KBS+18] that the errors of an ϵ-PIP may exhibit input data dependencies. For example, denoting the bitwise AND between $\boldsymbol{x}$ and $\boldsymbol{k}$ as $\boldsymbol{x} \cdot \boldsymbol{k}$ and the Hamming weight function as $\mathsf{HW}(.)$, it is known that the error probability of a serial implementation of ϵ-PIP based on frequency or voltage over-scaling decreases with $\mathsf{HW}(\boldsymbol{x} \cdot \boldsymbol{k})$. It has also been shown that such input dependencies are significantly reduced in the case of parallel ϵ-PIP architectures. Given the easier error control of serial ϵ-PIP architectures, this has led the authors of [KSD+20, KBS+18] to consider mixed (parallel then serial) ϵ-PIP implementations as a good tradeoff for secure LPPN instances. In this section, we complement these first analyses of LPPN physical defaults by analyzing the output data dependencies and possible structure of physical errors. For this purpose, we simulate an ASIC design that is essentially similar to the one in [KBS+18] and is comprised of three parts: an inner product architecture, a variable delay line and an error controller. By controlling the output sampling clock thanks to the variable delay line, this design allows accurately controlling the error rate. Implementation details are not necessary for the understanding of this section, but are given in Appendix A for completeness.

3.1 Output data dependencies of the errors and mitigation

In the LPN setting, the error $e \leftarrow \mathsf{Ber}_\epsilon$ is completely independent on the output of the inner product, since it is generated separately thanks to a Random number Generator (RNG) or Pseudo-Random Number Generator (PRNG). By contrast, in the LPPN setting, inexact computations are deployed to implement a noisy inner product with embedded errors. Since these errors are generated by sampling the output incorrectly, it is therefore natural to investigate whether the error probability depends on the output data.

3.1.1 Evaluation settings

Our goal in this first subsection is to show that LPPN samples can be affected by output dependencies, and that these dependencies can to some extent be mitigated thanks to adequate architectural choices. For this purpose, we next analyze various solutions

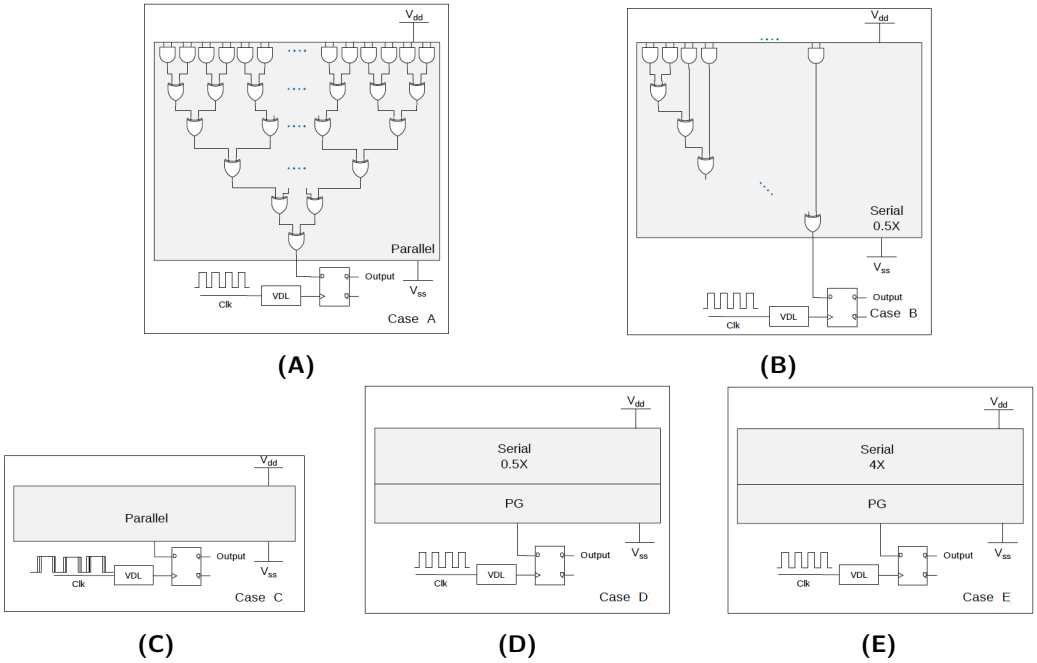


Figure 1: ϵ-PIP architectures.

to compute 32-bit LPPN samples and compare the output dependencies of their error probabilities.[3] The cases we consider are listed next and illustrated in Figure 1:

- Case A (parallel architecture). The 32-bit ϵ-PIP is implemented with a tree of depth 6 (1 AND stage & 5 XOR stages). The glitches are minimized since there are limited imbalances between the routing paths of parallel architecture. The variable delay line (used to control the output sampling clock) is fed with an ideal clock source.

- Case B (serial architecture). The 32-bit ϵ-PIP is implemented with 32 sequential stages (1 AND stage & 31 XOR stages). The amount of glitches increases with the logic depth of the combinatorial design because the imbalances between the routing paths are maximized. Minimum sized gates provided by the foundry are used (next denoted as 0.5X). The variable delay line is fed with an ideal clock source.

- Case C (parallel architecture with jitter). This is the same parallel architecture as described above (in case A), but with a jittery clock source feeding the variable delay line such that the output is sampled with some random inaccuracies. As a result, the errors are due to both deterministic and probabilistic (jitter) effects.

- Case D (serial architecture with power gating). This case uses the same serial architecture as described above (in case B) but we add some power gating cells to the ground net in order to reduce the current that drives a zero. It allows balancing the propagation time of the 0←1 and 1←0 transitions and implies a more balanced generation of glitches towards 0 or 1, aimed to reduce the output dependencies.

- Case E (serial architecture with balanced gates and power gating). This is the same as case D but we use bigger and more balanced gates (next denoted as 4X).

[3] A 32-bit architecture is chosen to limit the simulation time of the comparisons in this section. But a more practically-relevant 512-bit architecture will be analyzed in Section 6.

Our case studies were implemented in a 65nm TSMC CMOS technology using the regular voltage threshold (RVT) flavor. Pre-layout simulations were performed using ULTRASIM within the CADENCE environment. We used 10^4 uniformly distributed random 32-bit input $\boldsymbol{x}$ vectors and a single uniformly distributed random 32-bit secret key $\boldsymbol{k}$.[4] The supply voltage is fixed at nominal 1.2V and the clock frequency is 2MHz. For each case, an adapted variable delay line is designed in order to sample the output of the inner product incorrectly, aiming to achieve an error probability close to 0.25. In the parallel with jitter case, we used a clock source with 200ps RMS jitter (the latter refers to the standard deviation of the jitter's Gaussian distribution). Finally, we took advantage of the readily available power gating standard cells of the 65nm TSMC CMOS technology in order to implement the serial architecture with power gating (i.e., case D).

In our evaluations, we estimate the error probability of our ϵ-PIPs in two parts. First the error probability ϵ_0 if $\langle \boldsymbol{x}, \boldsymbol{k} \rangle = 0$, second the error probability ϵ_1 if $\langle \boldsymbol{x}, \boldsymbol{k} \rangle = 1$. Then, we compute the following normalized difference between the error probabilities as:

$$\Delta = \frac{\left| \hat{\Pr}[e \leftarrow \epsilon_0\text{-PIP}] - \hat{\Pr}[e \leftarrow \epsilon_1\text{-PIP}] \right|}{2}.$$

3.1.2 Output data dependencies results

The output data dependencies captured by Δ for all the case studies are shown in Figure 2 and lead to a number of useful observations that we list next:

1. The error probability of the parallel architecture (case A) shows significant data dependencies ($\Delta \approx 4.2\%$). Their main explanation is that the errors in this case are mostly generated by the incorrect sampling of the previous correct output bit by an ideal clock source. In other words, these are structured errors (as will be discussed in the next subsection) and they are mostly deterministic, since they depend on the propagated data. The fact that the first stage of the ϵ-PIP is an AND then favors an output zero rather than one. Hence, the probability of having an error given that the correct output is one is higher than in the opposite case, leading to an imbalance.

2. The errors of the serial architecture (case B) suffer from significant data dependencies as well ($\Delta \approx 3.7\%$). However, the reason is quite different than in the parallel case. In the serial architecture the errors are generated by sampling the deterministic glitches. We posit that imbalanced glitches are in cause in this case. Indeed, we used minimum sized gates that provide different propagation times in case the gate output goes from high to low or the inverse. More precisely, these gates switch their outputs faster to zero than they do to one. As a result, we observe a higher probability of error given that the correct output is one than in the opposite case.

3. Adding a jittery clock source to the parallel architecture (case C) is a nice solution to reduce the errors' data dependencies below 1%. Intuitively, a jittery clock implies more probabilistic (random) inaccuracies in the sampling process of the output. As a result, even though the error is still the result of sampling the previous correct bit, the use of a jittery clock masks the data dependency to a good extent.

4. As an attempt to mitigate the errors' data dependencies for the serial architecture, we considered a gate-level solution that reduces the current that drives a zero by means of power gating cells applied only to the ground net (case D). As it is clear from Figure 2, this proposal reduced the errors' data dependencies ($\Delta \approx 1.6\%$).

[4] Since the calibration phase to set the error control is adapted to the key (see Appendix A), the conclusions in this section are not significantly affected by the value of this key.

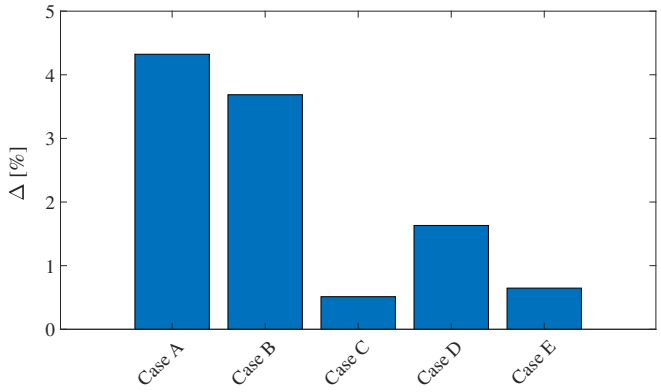


Figure 2: Output data dependencies of the probability of error of different ϵ-PIP architectures. Case A: parallel architecture. Case B: serial architecture with minimum sized gates. Case C: parallel architecture with jitter. Case D: serial architecture with minimum sized gates and power gating. Case E: serial architecture with bigger gates and power gating.

5. Finally, we explored differently sized gates that have more driving strength (therefore are bigger) and more balanced propagation time (i.e., the time needed for the gate's output to go from high to low is close to the time it needs to go from low to high) along with power gating applied to the ground net (case E). Clearly, this solution significantly reduces the errors' data dependencies so that Δ is below 1%.

We conclude from these preliminary simulations that significant output dependencies can be observed in LPPN samples. These dependencies can be reduced thanks to ASIC design tweaks and their detailed analysis and improvement is an interesting scope for further research. However, it is unlikely to cancel such physical defaults completely. We use this result as an important motivation for analyzing these dependencies and proving that they do not cancel the hardness of the LPPN problem in Section 4.

3.2 Structured errors in parallel implementations and mitigation

Besides the output data dependencies of the error distribution, another physical default which is quite specific to parallel ϵ-PIPs is the fact that LPPN errors can be structured. More precisely, when two consecutive (parallel) inner products are computed, namely $\langle \boldsymbol{x1}, \boldsymbol{k} \rangle$ and $\langle \boldsymbol{x2}, \boldsymbol{k} \rangle$, the result of the second inner product can either be correct (i.e., equal to $\langle \boldsymbol{x2}, \boldsymbol{k} \rangle$) or incorrect and equal to the previous correct value $\langle \boldsymbol{x1}, \boldsymbol{k} \rangle$ (but wether it is one or the other option that is observed remains unknow to the adversary).

To demonstrate this physical default, we estimated the probability that the second error corresponds to the correct output value given by the first inner product computation, that is $\Pr[e_2 = \langle \boldsymbol{x1}, \boldsymbol{k} \rangle]$, for both the parallel (A) case and serial (B) case of the previous subsection. For the parallel implementation, this probability was worth 0.98 while it was of 0.53 for the serial one. The (much) less structured errors of the serial implementation are explained by the fact that errors are then sampled from glitches and therefore, there is a nearly balanced chance that the glitch is equal to the previous correct output.

We note that such structured errors are easy to prevent at the design level. One option is to output one every two inner product computations (which halves the throughput). This way the neglected output conceals the previous correct one. A more efficient solution would be to use a dual-edge clock implementation, where the inputs are sampled at both the rising and falling edges of the clock, and only considering the output due to one of the clock

edges. Most interestingly, this problem also vanishes in case we mask the implementation since inner product computations are randomized in this case. Since structured errors are quite specific to parallel implementations and easy to fix by design, we do not formalize them further and next focus on the more critical issue of output data-dependent errors.

4 Reduction between LPN and LPPN

The previous section showed that the ϵ-PIPs generating LPPN samples may be affected by physical defaults that can create output dependencies in the error distribution. Due to their physical nature, it is unlikely to completely cancel these dependencies at the design level. In this section, we therefore aim to demonstrate that despite these dependencies, the security provided by the corresponding LPPN samples does not fundamentally differ from the security of LPN samples. For this purpose, and as usual when reasoning about physical primitives, we face a trade-off between the generality of security claims and their practical relevance. That is, in theory, there are as many LPPN instances with output dependencies as there are physical functions. But providing a separate formal treatment for each physical function is not possible. We deal with this issue by specifying a family of LPN problems that covers classes of imperfections that may be observed in practice, which we denote as the LPN with Output Dependencies (LPN-OD) problem. We then show a self-reduction of the LPN-OD problem, which gives in particular an equivalence between the LPN-OD problem and the standard LPN problem, with given parameters. The equivalence ensures the intractability of LPN-OD from the one of LPN. We show how to use it to estimate parameters for our physical problem at the end of the section.

4.1 Notations

We first define the LPN problem with output error dependencies as follows:

Definition 5 (LPN problem with output dependencies)**.** Let $\langle ., . \rangle$ denote the binary inner product, $\boldsymbol{k}$ be a random n-bit secret, $\epsilon \in [0, \frac{1}{2}]$ be a noise parameter, Ber_ϵ be the Bernoulli distribution with parameter ϵ and $D_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$ be the distribution defined as:

$$D_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1} =: \{(\boldsymbol{x}, \langle \boldsymbol{x}, \boldsymbol{k} \rangle \oplus e) : \boldsymbol{x} \leftarrow \{0,1\}^n; e \leftarrow \mathsf{Ber}_{\epsilon_0} \text{ if } \langle \boldsymbol{x}, \boldsymbol{k} \rangle = 0, e \leftarrow \mathsf{Ber}_{\epsilon_1} \text{ otherwise}\}.$$

Let $\mathcal{O}_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$ denote an oracle outputting independent samples according to the $D_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$. The LPN-OD$_{\epsilon_0,\epsilon_1}^n$ problem is said to be (q, t, m, θ)-hard to solve if for any algorithm A,

$$\Pr[\boldsymbol{k} \leftarrow \{0,1\}^n : A^{\mathcal{O}_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}}(1^n) = \boldsymbol{k}] \leq \theta,$$

and A runs in time $< t$, with memory $< m$ and makes at most q queries to $\mathcal{O}_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$.

Proposition 1 (LPN-OD alternative characterization)**.** *Let $n \in \mathbb{N}, \boldsymbol{k} \in \mathbb{F}_2^n \setminus \{0\}, \epsilon_0, \epsilon_1 \in [0, 1/2]$. Samples $(\boldsymbol{a}, b) \in \mathbb{F}_2^n \times \mathbb{F}_2$ follow $D_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$ if and only if $\boldsymbol{a}$ follows a uniform distribution and the following equations hold:*

$$\begin{cases} Pr[b = 1] = \dfrac{1}{2} + \dfrac{\epsilon_0 - \epsilon_1}{2}, \\ Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 0] = \dfrac{1 - \epsilon_0}{1 - \epsilon_0 + \epsilon_1}, \\ Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1] = \dfrac{\epsilon_0}{1 + \epsilon_0 - \epsilon_1}. \end{cases}$$

Proof. Let $(\boldsymbol{a}, b) \in \mathbb{F}_2^n \times \mathbb{F}_2$ be sampled from $D_{\boldsymbol{k}}^{\epsilon_0,\epsilon_1}$ with $b = \langle \boldsymbol{a}, \boldsymbol{k} \rangle + e$.

First, we prove that, for $b \in \{0,1\}, \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = b] = \frac{1}{2}$. Let $j = \min\{i \in [n] \,|\, \boldsymbol{k}_i = 1\}$ (j is well defined since $\boldsymbol{k} \neq 0$), therefore $\langle \boldsymbol{a}, \boldsymbol{k} \rangle = \left(\sum_{i \neq j} \boldsymbol{a}_i \boldsymbol{k}_i \right) + \boldsymbol{a}_j$. Since $\boldsymbol{a}_j$ is uniformly random over $\mathbb{F}_2$ and $b \in \{0,1\}$, it gives $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = b] = \frac{1}{2}$. We can then show:

$$\Pr[b=1] = \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \cap e = 1] + \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 1 \cap e = 0] = \frac{1}{2}(\epsilon_0 + 1 - \epsilon_1) = \frac{1}{2} + \frac{\epsilon_0 - \epsilon_1}{2},$$

$$\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 0] = \Pr[b = 0 \,|\, \langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0] \frac{\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0]}{\Pr[b=0]} = \frac{1 - \epsilon_0}{1 - \epsilon_0 + \epsilon_1},$$

$$\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1] = \Pr[b = 1 \,|\, \langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0] \frac{\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0]}{\Pr[b=1]} = \frac{\epsilon_1}{1 + \epsilon_0 - \epsilon_1}.$$

For the reverse implication, since $\boldsymbol{a}$ is uniformly distributed and the three probabilities $\Pr[b=1]$, $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1]$ and $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1]$ uniquely define the pair $\Pr[b = 1 \,|\, \langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0] = \epsilon_0$ and $\Pr[b = 0 \,|\, \langle \boldsymbol{a}, \boldsymbol{k} \rangle = 1] = \epsilon_1$, it allows to conclude. □

Remark. LPN-OD is a more general problem than LPN. Indeed, for $\epsilon_0 = \epsilon_1$, $\text{LPN-OD}^n_{\epsilon_0,\epsilon_1}(\boldsymbol{k})$ is equivalent to $\text{LPN}^n_{\epsilon_0}(\boldsymbol{k})$. Besides, the two problems defined above are so-called *search* problems, where the goal of the adversary is to recover $\boldsymbol{k}$. One could also define a *decision* version of these two problems, that will not be studied in this paper.

4.2 Reduction

We next show that the standard LPN (resp., LPN-OD) problem is as hard as the LPN-OD (resp., standard LPN) problem and there is both a polynomial reduction from LPN-OD to LPN and from LPN to LPN-OD. To do so, we use a self-reduction of the LPN-OD problem and show that we can transform in polynomial time an instance of LPN-OD into a noisier one with the same key and a possibly different balance between the two noise parameters. Precisely, we show that a probabilistic polynomial time algorithm can produce an instance of $\text{LPN-OD}^n_{\epsilon'_0,\epsilon'_1}(\boldsymbol{k})$ from an instance of $\text{LPN-OD}^n_{\epsilon_0,\epsilon_1}(\boldsymbol{k})$ (for noise parameters satisfying restrictions specified later). Fixing $\epsilon'_0 = \epsilon'_1$ gives a reduction from LPN-OD to LPN (breaking LPN breaks LPN-OD) and fixing $\epsilon_0 = \epsilon_1$ gives a reduction from LPN to LPN-OD (breaking LPN-OD breaks LPN). The algorithm has access to an oracle sampling $D^{\epsilon_0,\epsilon_1}_{\boldsymbol{k}}$ with ϵ_0 and ϵ_1 known, and the reduction is based on two main ideas: the control of the (un)-balancedness of the b part, and the addition of two Bernouilli distributions.

First, as seen in Proposition 1, the probability $\Pr[b=1]$ depends on the noise parameters. Hence, the algorithm uses a method of rejection sampling in order to produce samples with b having the target probability of samples from $D^{\epsilon'_0,\epsilon'_1}_{\boldsymbol{k}}$. Second, a standard idea to turn LPN samples into noisier samples in a black-box manner is to add a Bernoulli on the b component. Using this idea, LPN-OD samples have two noise parameters linked with the value of the correct inner product computation and the algorithm acts differently on these two parameters, adding two Bernouilli's next denoted as t_0 and t_1, and choosing which one to add based on the value given by the rejection sampling.

We first let $\text{SampleLPN-OD}^n_{\epsilon_0,\epsilon_1}(\boldsymbol{k})$ describe the oracle $\mathcal{O}^{\epsilon_0,\epsilon_1}_{\boldsymbol{k}}$ (from Definition 5) that returns samples from the distribution $D^{\epsilon_0,\epsilon_1}_{\boldsymbol{k}}$ for a key $\boldsymbol{k}$ in Algorithm 1.

We then use this sampler in Algorithm 2. The idea is to start by fixing the b of the output in order to ensure that it follows the good distribution. We then control the output probabilities by adding different Bernoulli errors t_0, t_1 depending on the fixed value of b. The correctness of our algorithm is given with the following lemma:

Algorithm 1: SampleLPN-OD$^{n}_{\epsilon_0,\epsilon_1}(\boldsymbol{k})$

Output: A LPN-OD sample $(\boldsymbol{a}, b)$
$\boldsymbol{a} \xleftarrow{\$} \mathbb{F}_2^n, e \xleftarrow{\$} \mathsf{Ber}_{\epsilon_i}$ where $i = \langle \boldsymbol{a}, \boldsymbol{k} \rangle$;
$(\boldsymbol{a}, b) \leftarrow (\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{k} \rangle + e)$;
return $(\boldsymbol{a}, b)$

Algorithm 2: AltSampleLPN-OD$^{n}_{\epsilon_0,\epsilon_1,t_0,t_1,m}(\boldsymbol{k})$

Input: A sampling upper limit $m \in \mathbb{N}$, error parameters $\epsilon_0, \epsilon_1 \in \left[0, \frac{1}{2}\right]$, Bernoulli parameters $t_0, t_1 \in \left[0, \frac{1}{2}\right]$ with $t_0 + t_1 \neq 1$
Output: A sample $(\boldsymbol{a}, b) \in \mathbb{F}_2^n \times \mathbb{F}_2$ following the distribution $D_{\boldsymbol{k}}^{\epsilon_0', \epsilon_1'}$
$\epsilon_0' = \frac{2\,\epsilon_0 t_0 - (2\,(2\,\epsilon_0 - 1) t_0 - 2\,\epsilon_0 + 1) t_1 - \epsilon_0}{t_0 + t_1 - 1}$;
$\epsilon_1' = \frac{(2\,\epsilon_1 - 1) t_0 - 2\,((2\,\epsilon_1 - 1) t_0 - \epsilon_1) t_1 - \epsilon_1}{t_0 + t_1 - 1}$;
if $\epsilon_0' \notin \left[0, \frac{1}{2}\right]$ *or* $\epsilon_1' \notin \left[0, \frac{1}{2}\right]$ **then**
  return $\perp_0$
end
$b' \xleftarrow{\$} \mathsf{Ber}_{\frac{1}{2} + \frac{\epsilon_0' - \epsilon_1'}{2}}$;
cnt := 0;
do
  $(\boldsymbol{a}, b'') \xleftarrow{\$}$ SampleLPN-OD$^{n}_{\epsilon_0,\epsilon_1}(\boldsymbol{k})$;
  $e' \xleftarrow{\$} \mathsf{Ber}_{t_{b'}}$;
  $(\boldsymbol{a}, b) \leftarrow (\boldsymbol{a}, b'' + e')$;
  cnt $\leftarrow$ cnt +1;
while $b \neq b'$ *and* $cnt \leq m$;
if $cnt = m$ **then**
  return $\perp$
end
return $(\boldsymbol{a}, b)$

Lemma 1. *For any non-zero* $\boldsymbol{k} \in \mathbb{F}_2^n$, $m \in \mathbb{N}$, *error parameters* $\epsilon_0, \epsilon_1 \in \left[0, \frac{1}{2}\right]$, $t_0, t_1 \in \left[0, \frac{1}{2}\right]$ *such that* $t_0 + t_1 \neq 1$, *if both* $\begin{cases} \epsilon_0' = \frac{2\,(\epsilon_0 (2\,t_1 - 1) - t_1) t_0 - \epsilon_0 (2\,t_1 - 1) + t_1}{1 - t_0 - t_1} \\ \epsilon_1' = \frac{(2\,\epsilon_1 (2\,t_1 - 1) - 2\,t_1 + 1) t_0 - \epsilon_1 (2\,t_1 - 1)}{1 - t_0 - t_1} \end{cases}$ *remain in* $\left[0, \frac{1}{2}\right]$, *then the output of AltSampleLPN-OD*$^{n}_{\epsilon_0,\epsilon_1,t_0,t_1,m}(\boldsymbol{k})$ *follows the distribution* $D_{\boldsymbol{k}}^{\epsilon_0', \epsilon_1'}$, *which is the same distribution as SampleLPN-OD*$^{n}_{\epsilon_0', \epsilon_1'}(\boldsymbol{k})$*'s output.*

Proof. We study the sample output probabilities of algorithm 2 and use Proposition 1 to show that it follows the distribution $D_{\boldsymbol{k}}^{\epsilon_0', \epsilon_1'}$. First, note that $\boldsymbol{a}$ is chosen uniformly. We aim at comparing the output probabilities with those of a regular sampler.

Keeping the notations of the algorithm, we have that $\Pr[b = 1] = \Pr[b' = 1] = \frac{1}{2} + \frac{\epsilon_0' - \epsilon_1'}{2}$. Therefore, it verifies the first equality of Proposition 1.

We now need to express $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \mid b = 0]$ and $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \mid b = 1]$ with the inputs and parameters of Algorithm 2: $\epsilon_0, \epsilon_1, t_0, t_1$. Note that the sum of two Bernoulli variables of parameters ϵ, ϵ' is a Bernoulli variable of parameter $\epsilon + \epsilon' - 2\epsilon\epsilon'$. With that in mind, for $i, j \in \{0, 1\}$, we introduce new error parameters $E_{i,j} = \epsilon_i + t_j - 2\epsilon_i t_j$. Since $b = \langle \boldsymbol{a}, \boldsymbol{k} \rangle + e + e'$ is computed as the sum of a LPN-OD sample and a Bernoulli, where $e \xleftarrow{\$} \mathsf{Ber}_{\epsilon_{\langle \boldsymbol{a}, \boldsymbol{k} \rangle}}$ and $e' \xleftarrow{\$} \mathsf{Ber}_{t_{b'}}$, and inside the loop, $(\boldsymbol{a}, b)$ follows an independent distribution $D_{\boldsymbol{k}}^{E_{0,b'}, E_{1,b'}}$, applying

Proposition 1 gives us $\begin{cases} \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 0] = \frac{1-E_{0,b'}}{1-E_{0,b'}+E_{1,b'}} \\ \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1] = \frac{E_{0,b'}}{1+E_{0,b'}-E_{1,b'}} \end{cases}$. After the loop condition filtering $b = b'$, for $i \in \{0,1\}$, we get $\Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = i] = \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b' = i]$, and:

$$\begin{cases} \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 0] = \frac{1-E_{0,0}}{1-E_{0,0}+E_{1,0}}, \\ \Pr[\langle \boldsymbol{a}, \boldsymbol{k} \rangle = 0 \,|\, b = 1] = \frac{E_{0,1}}{1+E_{0,1}-E_{1,1}}. \end{cases}$$

We can now put in relation those three equations with the ones given by Proposition 1 for the output of a LPN-OD sampler (Algorithm 1):

$$\begin{cases} \frac{1-E_{0,0}}{1-E_{0,0}+E_{1,0}} = \frac{1-\epsilon'_0}{1-\epsilon'_0+\epsilon'_1} \\ \frac{E_{0,1}}{1+E_{0,1}-E_{1,1}} = \frac{\epsilon'_0}{1+\epsilon'_0-\epsilon'_1} \end{cases} \iff \begin{cases} \epsilon'_0 = \frac{2(\epsilon_0(2t_1-1)-t_1)t_0-\epsilon_0(2t_1-1)+t_1}{1-t_0-t_1}, \\ \epsilon'_1 = \frac{(2\epsilon_1(2t_1-1)-2t_1+1)t_0-\epsilon_1(2t_1-1)}{1-t_0-t_1}. \end{cases}$$

The equivalence allows us to conclude that the output of $\text{AltSampleLPN-OD}^n_{\epsilon_0,\epsilon_1,t_0,t_1,m}(\boldsymbol{k})$ follows $D^{\epsilon'_0,\epsilon'_1}_{\boldsymbol{k}}$, the same distribution as $\text{SampleLPN-OD}^n_{\epsilon'_0,\epsilon'_1}(\boldsymbol{k})$'s output. □

Remark. Using the lemma, one can either:

- Use a LPN sampler to simulate an asymmetric LPN-OD one.
- Use a LPN-OD sampler to simulate a symmetric LPN one.

Theorem 1. *The LPN problem is polynomially equivalent to the LPN-OD problem. More precisely LPN^n_ϵ is exactly $\text{LPN-OD}^n_{\epsilon,\epsilon}$, and there is a polynomial reduction from $\text{LPN-OD}^n_{\epsilon_0,\epsilon_1}$ to $\text{LPN-OD}^n_{\epsilon'_0,\epsilon'_1}$ for all $\epsilon_0, \epsilon_1, \epsilon'_0, \epsilon'_1$ verifying the equations given by Lemma 1.*

Proof. From Lemma 1, Algorithm 2 allows to simulate $D^{\epsilon'_0,\epsilon'_1}_{\boldsymbol{k}}$ from $D^{\epsilon_0,\epsilon_1}_{\boldsymbol{k}}$ with the appropriate parameters. Hence, we must show that, with m polynomial in n, Algorithm 2 is a probabilistic polynomial time algorithm, and it returns $\perp$ with negligible probability.

First, note that Lemma 1 requires $\epsilon'_0, \epsilon'_1 \in \left[0, \frac{1}{2}\right]$. Hence, the probability of returning $\perp_0$ is 0. Then, since the number of calls to SampleLPN is bounded by m and the sampler Algorithm 1 runs in constant time, for m polynomial in n the execution time is polynomial in n. The algorithm returns $\perp$ only if the samples given by the sampler have the wrong value of b for m consecutive trials, hence we can study this probability using a binomial law. The probability of returning $\perp$ is $\Pr[b' = 1] \cdot \Pr[b = 0 \,|\, b' = 1]^m + \Pr[b' = 0] \cdot \Pr[b = 1 \,|\, b' = 0]^m$. Let $i \in \{0,1\}$, we show that the probabilities $\Pr[b' = i]$ and $\Pr[b = i \,|\, b' = 1-i]$ always belong to $[\frac{1}{4}, \frac{3}{4}]$. The value of b' is given by a Bernouilli of parameter $\frac{1}{2} + \frac{\epsilon'_0-\epsilon'_1}{2}$ where $\epsilon'_0, \epsilon'_1 \in [0, \frac{1}{2}]$. Based on the analysis in the proof of Lemma 1, $\Pr[b = i \,|\, b' = 1-i]$ is given by a Bernouilli of parameter $\frac{1}{2} + \frac{E_{0,b'}-E_{1,b'}}{2}$, and we show that $E_{0,b'}, E_{1,b'}$ are also in $[0, \frac{1}{2}]$. These quantities are obtained as $x + y - 2xy$ where $x, y \in [0, \frac{1}{2}]$. Fixing y, we define the function $f_y(x)$ on $[0, \frac{1}{2}]$ as $f_y(x) = x(1-2y) + y$, its derivative $(1-2y)$ is positive for $y \neq \frac{1}{2}$ and null otherwise. Hence the maximum of $f_y(x)$ is reached in $\frac{1}{2}$, and the minimum in 0. It allows concluding that $0 \leq x + y - 2xy \leq \frac{1}{2}$. Finally, the probability of failure is at most $2\left(\frac{3}{4}\right)^{m+1}$, therefore negligible in n. □

4.3 Concrete security estimation

Theorem 1 gives a general self-reduction of the LPN-OD problem, encompassing the equivalence between LPN-OD and LPN. But in order to quantify the security we can expect from the real samples we obtain from our prototype LPPN processor in Section 6.4, we only need the reduction from LPN-OD to LPN. For this purpose, we simply consider that these samples are the ones of a LPN-OD problem (Definition 5) and we determine an

instance of a classical LPN problem which can be converted to it. Since we can assume that an adversary recovering the key from LPPN samples could break $\mathrm{LPN}_\epsilon^n(\boldsymbol{k})$, we can estimate the security of the instances of Section 6.4 with the complexity of the best known attacks on LPN with a noise parameter ϵ (assuming the polynomial reduction has a negligible cost compared to this complexity). This (LPN-OD to LPN) reduction is a particular case of Theorem 1. We state it in the following corollary. This result shows that for noise parameters of LPN-OD centered around the same value, the less these two parameters differ, the higher is the parameter of the LPN problem it reduces to.

Corollary 1. *For any $n, m > 1$, non-zero $\boldsymbol{k} \in \mathbb{F}_2^n$, ϵ' and Δ error parameters such that $0 \leq \epsilon' - \Delta \leq \epsilon' + \Delta \leq \frac{1}{2}$, if an adversary is able to retrieve the key $\boldsymbol{k}$ given m samples from $\text{LPN-OD}_{\epsilon'-\Delta,\epsilon'+\Delta}^n(\boldsymbol{k})$ then a polynomially equivalent adversary can retrieve the key from $\mathrm{LPN}_\epsilon^n(\boldsymbol{k})$ using m samples, where $\epsilon = \frac{\epsilon'-\Delta}{1-2\Delta}$.*

Proof. It corresponds to the particular case where the adversary uses the polynomial reduction from $\text{LPN-OD}_{\epsilon_0,\epsilon_1}^n$ to $\text{LPN-OD}_{\epsilon_0',\epsilon_1'}^n$ with $\epsilon_0 = \epsilon_1 = \frac{\epsilon'-\Delta}{1-2\Delta}$, $\epsilon_0' = \epsilon' - \Delta$ and $\epsilon_1' = \epsilon' + \Delta$. We check if these values of $\epsilon_0, \epsilon_1, \epsilon_0'$ and ϵ_1' verify the equations of Lemma 1. The relations $0 \leq \epsilon' - \Delta \leq \epsilon' + \Delta \leq \frac{1}{2}$ give $\epsilon' \in [0, \frac{1}{2}]$ and $\Delta \in [0, \frac{1}{2}[$, hence $\epsilon_0, \epsilon_1 \in [0, \frac{1}{2}]$. Then, taking $t_0 = \frac{2\Delta}{2\Delta+1}$ and $t_1 = 0$ we get $t_0, t_1 \in [0, \frac{1}{2}]$, $t_0 + t_1 \neq 1$ and:

$$\begin{cases} \epsilon_0' = \frac{2\,(\epsilon_0(2\,t_1-1)-t_1)t_0-\epsilon_0(2\,t_1-1)+t_1}{1-t_0-t_1} = \epsilon' - \Delta, \\ \epsilon_1' = \frac{(2\,\epsilon_1(2\,t_1-1)-2\,t_1+1)t_0-\epsilon_1(2\,t_1-1)}{1-t_0-t_1} = \epsilon' + \Delta, \end{cases}$$

$\epsilon' \pm \Delta \in [0, \frac{1}{2}]$ hence the 4 parameters verify the equations. In conclusion, Theorem 1 gives a polynomial reduction from $\mathrm{LPN}_\epsilon^n(\boldsymbol{k})$ to $\text{LPN-OD}_{\epsilon'-\Delta,\epsilon'+\Delta}^n(\boldsymbol{k})$. Since the reduction conserves the number of samples, m, it gives the final result: an adversary breaking $\text{LPN-OD}_{\epsilon'-\Delta,\epsilon'+\Delta}^n(\boldsymbol{k})$ with m samples implies the existence of a (polynomially equivalent) adversary breaking $\mathrm{LPN}_\epsilon^n(\boldsymbol{k})$ with m samples. □

Remark. By ensuring a smaller output unbalancing parameter Δ (e.g., thanks to implementation tweaks), we manage to reduce the LPN-OD problem to a stronger LPN problem (with higher parameter). Indeed, for all $\epsilon' \in \left[0, \frac{1}{2}\right]$, we can introduce the function $f_{\epsilon'}$ defined for $\Delta \in \left[0, \frac{1}{2}\right[$ as $f_{\epsilon'}(\Delta) = \frac{\epsilon'-\Delta}{1-2\Delta}$. This function returns the parameters of the LPN problem our LPN-OD instance reduces to. Its derivative is such that $f'_{\epsilon'}(\Delta) = -\frac{1-2\epsilon'}{(1-2\Delta)^2} \leq 0$, which means that f increases when Δ decreases. Accordingly, the smaller the Δ, the bigger the ϵ, and the more secure the corresponding LPN problem is. Note that a similar corollary applies for $\text{LPN-OD}_{\epsilon'+\Delta,\epsilon'-\Delta}^n$ and LPN_ϵ^n, by inverting the roles of t_0 and t_1 in the proof.

5 Masked LPPN

The previous section showed that the LPN-OD problem is hard. Yet, it also suggests that its hardness increases if designers of LPPN-based implementations are able to limit the output data dependencies of the error probability. In this section, we briefly discuss how much masking an ϵ-PIP can help to mitigate such data dependencies. Admittedly, the high-level idea behind our observations was already given in [KSD+20], and we only specialize it to LPN-OD. In view of the general difficulty to model physical leakages, our following discussion is not based on security reductions but on attack-based arguments. We leave their formalization as an open problem and use it to argue that masking can significantly mitigate "filtering attacks" exploiting output-dependent errors.

Our starting observation for this purpose is the following one. Imagine that an ϵ-PIP is implemented in a masked manner. That is, rather than computing $y = \epsilon\text{-PIP}(\boldsymbol{x}, \boldsymbol{k})$ in

an incorrect manner, one would compute $y = \epsilon\text{-PIP}(\boldsymbol{x}, \boldsymbol{k}_1) + \langle \boldsymbol{x}, \boldsymbol{k_2} \rangle + \ldots + \langle \boldsymbol{x}, \boldsymbol{k}_d \rangle$, where $d-1$ shares $\boldsymbol{k}_1, \boldsymbol{k}_2, \ldots, \boldsymbol{k}_{d-1}$ are picked up uniformly at random and $\boldsymbol{k} = \boldsymbol{k}_1 + \boldsymbol{k}_2 + \ldots + \boldsymbol{k}_d$. In this case, an adversary willing to filter the (incorrect) outputs in order to take advantage of the data-dependent errors would not have access to the output of the ϵ-PIP causing the errors, let's denote it as $z = \epsilon\text{-PIP}(\boldsymbol{x}, \boldsymbol{k}_1)$, but only to its leakage, denoted as $\boldsymbol{l}$.

Besides the fact that such a masking improves security against Differential Power Analysis (DPA), as discussed in [KBBS20], it also reduces the Δ of the corresponding LPN-OD problem as follows. First, assume that the incorrect computation of z indeed suffers from output data dependencies of the error probability such that $\Pr[e = 1 | \langle \boldsymbol{x}, \boldsymbol{k}_1 \rangle = 0] = \epsilon_0$ and $\Pr[e = 1 \,|\, \langle \boldsymbol{x}, \boldsymbol{k}_1 \rangle = 1] = \epsilon_1$. Using the notations of Section 3, it means that $\epsilon_0 = \epsilon + \Delta$ and $\epsilon_1 = \epsilon - \Delta$ so that $|\epsilon_0 - \epsilon_1| = 2\Delta$. Then assume that the leakage does only allow recovering z with a certain probability given by $\Pr[z = 0 \,|\, \boldsymbol{l}] = \frac{1}{2} + \delta$ and $\Pr[z = 1 \,|\, \boldsymbol{l}] = \frac{1}{2} + \delta$.[5]

As a result, we directly have that the actual imbalance between the probabilities of error that the adversary will be able to exploit to filter is given by:

$$\epsilon'_0 = \epsilon_0 \cdot \left(\frac{1}{2} + \delta\right) + \epsilon_1 \cdot \left(\frac{1}{2} - \delta\right),$$
$$\epsilon'_1 = \epsilon_1 \cdot \left(\frac{1}{2} + \delta\right) + \epsilon_0 \cdot \left(\frac{1}{2} - \delta\right).$$

If $\delta = \frac{1}{2}$, then the recovery of z is perfect so that we have $\epsilon'_0 = \epsilon_0$ and $\epsilon'_1 = \epsilon_1$. If $\delta = 0$, then there is no leakage so that we have $\epsilon'_0 = \epsilon'_1 = \epsilon$. So the latter shows that in case an implementation is masked and the adversary does not exploit the leakage, then exploiting the output data dependencies with a filtering attack is not possible.

Quite naturally, the situation when an adversary exploits the leakage will fall in between (i.e., actual δ values will be between 0 and $\frac{1}{2}$). But it is important to see that the secret z to recover is an ephemeral one, so this adversary will be limited to Simple Power Analysis (SPA). We will discuss concrete values in Section 6. Note that the proposal in this section is aimed at hiding the output error dependencies, since it needs to be addressed for current prototypes of LPPN processors. But in case input dependencies become critical, the same idea can be used to mask $\boldsymbol{x}$. Note also that the proposal in this section induces the errors on a single share, which is interesting since it maintains the level of control needed on the error probability equal to the one of an unprotected implementation. If the error was spread over multiple shares, its probability would be lower on each share (as per the piling up lemma), leading to a more challenging calibration of the error probability.[6]

6 FPGA prototype

In this section, we present a first FPGA implementation of LPPN processor, demonstrating feasibility on a cheap and reprogrammable commercial device. Integrating the LPPN idea within standard FPGA resources increases its applicability, making it more attractive for a broader range of applications where the cost factor and design time are critical. We first describe our design, then verify its functional correctness and follow by investigating the (input and output) dependencies of its error probabilities. A side-channel evaluation is deferred to Appendix C, since it does not bring significantly new insights compared to what was observed for ASIC prototypes. Details about the area cost of our prototype and and comparisons with classical RNG-based LPN are in Appendix D. We conclude by discussing the security of our prototype based on the reduction of Section 4.

[5] As usual in side-channel analysis, we assume a leakage function with additive and data-independent noise so that the complexity of recovering z is the same whether it is zero or one.

[6] The side-channel based filtering attacks discussed in this section, which target the leakage of an ϵ-PIP's (unbalanced) incorrect output, are different from side-channel attacks targeting the correct output of LPN implementations, formalized in [DFH+16] with the Learning with Leakage (LPL) problem.

6.1 FPGA design

The proposed architecture of the LPPN processor is depicted in Figure 3A. The inner product block receives the input secret $\boldsymbol{k}$ and challenge $\boldsymbol{x}$ (both as 512-bit vectors) and implements an ϵ-PIP, by means of a specific sampling clock *clk_del*, generated by a Variable Delay Line (VDL) and having the role of producing a precise skew to obtain the desired (and inbound) probability of error. An *enable* signal is used to engage the LPPN processor to perform a single bit computation. The same *enable* signal is used as input for an on-chip voltage sensor, that provides an estimation of the working condition. As we will discuss later, the sensor is designed to detect changes in power supply voltage that may be related to the presence of static changes or voltage glitches during the inner product computation. Finally, a finite state machine handles the calibration of the VDL through a Successive Approximation Register (SAR) approach, based on the computation of the probability of error. In the following, we discuss these blocks more in detail.

Inner product block. The first investigations of LPPN prototypes (both the simulated ones in [KSD+20] and the tape out in [KBS+18]) pointed out that the probability of error of such prototypes exhibit dependencies in the Hamming weight of their input, which can be mitigated by using a mixed (parallel then serial) architecture for the inner product computations. Our FPGA implementation adopts the same architecture, which we depict in Figure 3B. The vectors $\boldsymbol{k}$ and $\boldsymbol{x}$ are initially multiplied in GF(2), which corresponds to a bitwise AND operation performed fully in parallel. The sum of this product is then implemented with a mix of parallel and serial XORs. More in detail, the parallel XOR tree receives 512 bits from the AND layer, that are XORed in parallel through six XOR layers (numbered from 1 to 6), providing an intermediate output sum vector of 8 bits. The output of the last layer of the parallel XOR stage is then forwarded as input to the serial XOR tree which recombines the 8 bits to the final sum of the inner product. The serial XOR net is made of 7 XOR gates, and since paths between its inputs and the final output are not balanced, the probability of generating glitches at this stage is very high.

The output bit P of the inner product net is sampled by two flip-flops in parallel. One is clocked by the properly skewed system clock, namely *clk_sk*, which always samples a stable value of the P signal, producing the correct output bit $P_{\text{out}}^{\text{corr}}$. A second flip-flop, in parallel with the first one, is clocked by a specifically skewed replica of the clock, namely *clk_del*. This flip-flop has the critical role to sample a metastable and/or glitched output of the inner product network. It is clear that to ensure the sampling of a non-stable P value, *clk_del* has to have less skew than *clk_sk*, as shown in Figure 3B-3C. By means of a specifically reduced skew, setup/hold violations may occur on this specific flip-flop, and, therefore, P_{out} represents the inexact computing of the inner product network.

Fully digital variable delay line. The key aspect to ensure that the LPPN processor is able to guarantee a bounded probability of error (and therefore, security level) is to design a control system and a VDL that are able to set the skew of *clk_del* to capture the non-stable state of the P signal. Possibilities to design a fine-grain VDL on FPGAs are reduced compared to ASICs, due to the limited degrees of freedom such off-the-shelf devices provide. Specifically, the ϵ-PIP needs fine granularity to provide a reliable control range over the probabilities of error. Post-place&route simulations performed on Xilinx ISim have suggested that the VDL has to be designed adopting a 2-part architecture: fixed delay part, needed to compensate the parallel XOR stage, and a variable delay part, to capture the glitches of the serial stage. To implement the fixed delay part, we have used LookUp Tables (LUTs) and routing delays, using 9 LUTs configured as buffers (providing ∼6ns of delay for our target FPGA). The FPGA implementation of the variable delay part of the VDL is based on the custom placing and configuration of CARRY4 chain primitives, as seen in other works such as [YRG+18, ZHL+13], along with a compact multiplexer

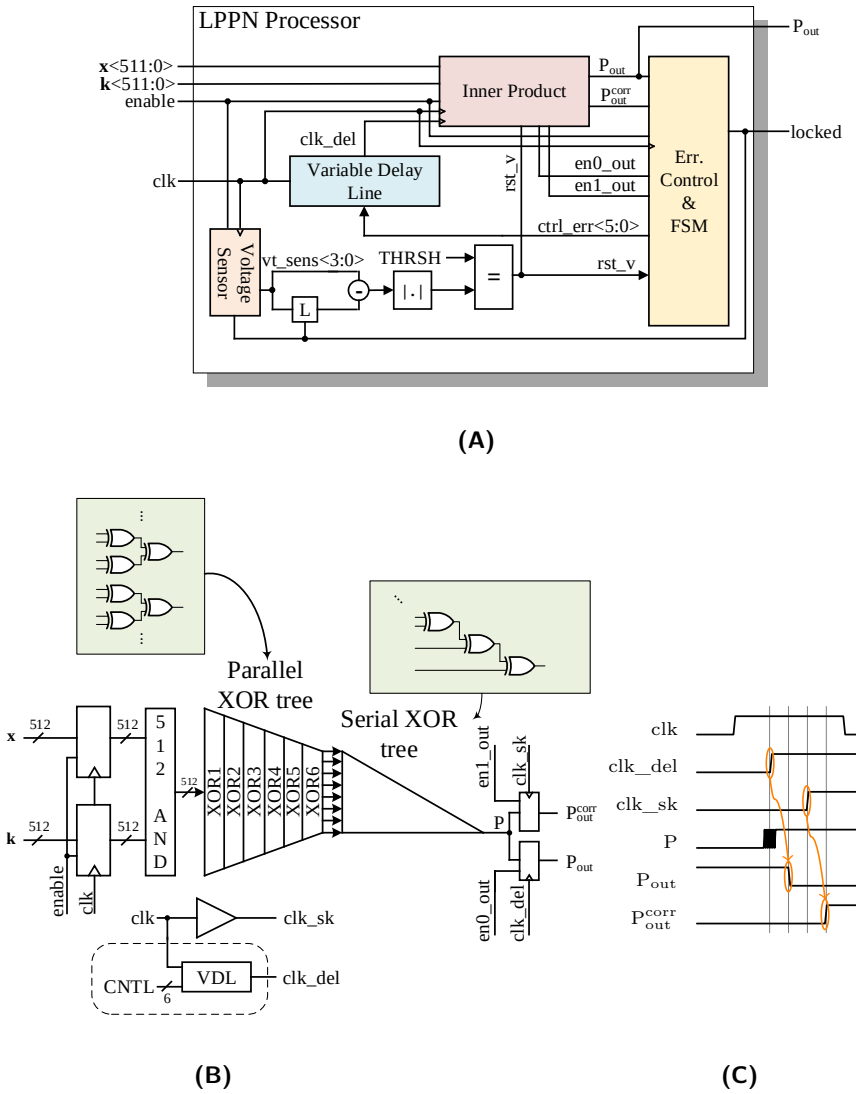


Figure 3: Architecture of the LPPN processor (A). Details of the inner product network (B) and corresponding timing diagram for inexact sampling (C).

built upon the SLICEL units. The CARRY4 primitives are widely used to implement high performance mathematical units, and their internal structure is designed to provide a fast path for carry bits propagation. In order to reach a tuning range of [0ns,∼2ns], we have placed 16 CARRY4 primitives, taking advantage of dedicated column routing to build our delay chain. We have obtained a total of 64 taps by using each CO output from all placed CARRY4 units. Post-place&route simulations have shown that each tap provides an average delay of 30ps. A compact 64:1 multiplexer has been designed according to the application note from Xilinx [Cha14], in vertically adjacent SLICEL sites, to route the selected tap to the dedicated flip-flop, in order to sample the output bit. This sub-block has been designed making use of 4:1 muxes in LUT6_L and in-slice MUXF7-MUXF8 primitives. This strategy allows ensuring a maximum usage of in-slice routing resources, thus minimizing delays and unpredictability. An additional LUT6 multiplexer has been used to generate the final output layer of our VDL, as shown in Figure 4.

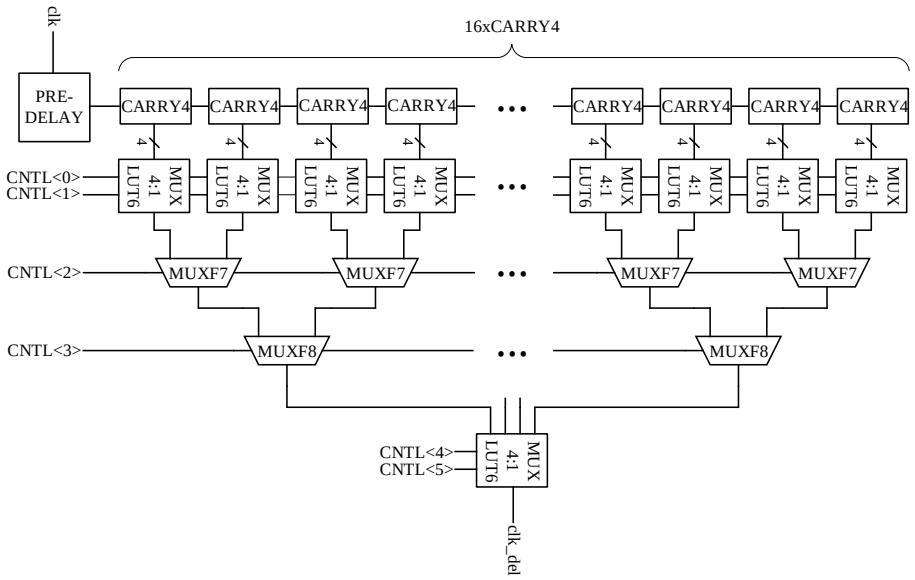


Figure 4: Architecture of the fully digital Variable Delay Line (VDL).

Fault detection scheme. To detect faulty working conditions during the authentication phase, we designed a fault detector based on an on-chip voltage sensor. The sensor has to be able to detect changes in the power supply voltage (since they are critical for the calibration of the processor) and the reaction to a positive detection has to be performed within 1 clock cycle. The sensor is implemented as a Time-to-Digital Converter (TDC) [UHO15] and its architecture is shown in Figure 5. It is constituted by a delay line of 93 LUTs, for which 8 non-evenly distributed taps are redirected to 8 flip-flops (sampled by the reference/nominal clock), in order to estimate a power supply voltage from 600mV up to 1.4V.[7] Each tap/bin of the resulting thermometric code represents approximately 100mV. A 0→1 transition of the *enable* signal triggers the sensor to perform a measurement. The higher the power supply, the faster is the transition of the *enable* signal across the delay line, as shown in Figure 5. As a result, more registers will consecutively sample a 1 at their input, providing a thermometric representation of the propagation delay/voltage measure. The thermometric-encoded output of the sensor is then converted to a 4-bit value, by means of a simple encoder. The output of the sensor is finally used to detect significant changes in the power supply voltage after the calibration phase. An additional register latches the output of the sensor at the end of the calibration, to use it as a reference. At the beginning of the computation of each requested inner product during the authentication phase, the sensor is triggered to produce a new measure, and the difference with the calibration value is computed. If the absolute value of the difference is higher than a certain threshold (e.g., 100mV worked best in our experiments), the LPPN processor is reinitialized. Consequently, the *locked* signal is reset to 0 (see Figure 3A) and a new calibration has to be performed. Using this approach, the LPPN processor is able to detect static changes (or even glitches) that an adversary could make (or inject) to alter the security of the LPPN processor.

Error control module. During the calibration phase, the LPPN processor computes 7 batches of 1024 challenge-secret inner products in order to set the variable delay line. During each batch, the Error Control module computes the XOR between the correct

[7] A uniform distribution of the taps would not be optimal for sampling since $V_{DD} \approx 1/t_{pd,LUT}$.

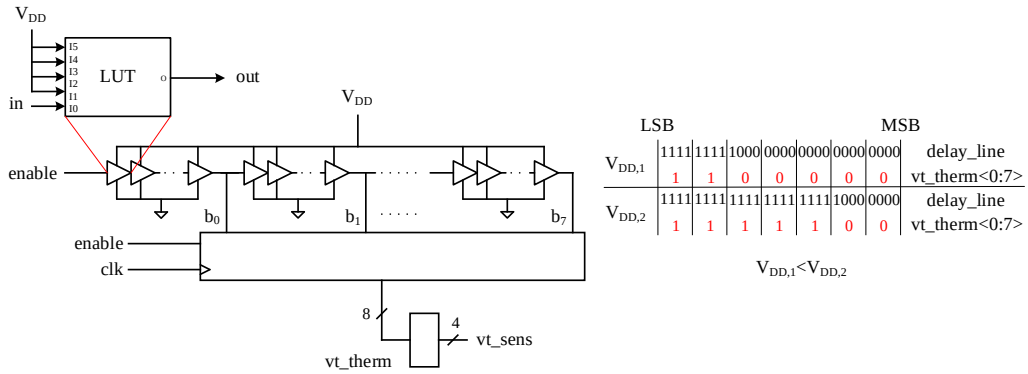


Figure 5: On-chip FPGA voltage sensor.

output P_{out}^{corr} with the inexact output P_{OUT} and it accumulates the result. At the end of each batch, the Error Control compares the accumulated results with a threshold (256, to eventually achieve $\Pr[e = 1] = 0.25$), and upon this comparison, it sets bit-by-bit, from the MSB to the LSB, the 6-bit $CNTL$ signal, used to drive the VDL. Concretely, the value of $CNTL$ is set to 0 at the beginning of the calibration. At the end of the 7^{th} batch, the LPPN is locked and the flip-flop sampling P_{out}^{corr} is disabled in order to reduce possible side-channel leaks. It has to be noted that in case of a positive detection of a faulty working condition, both the inner product and the error controller modules are reinitialized, and a new calibration is therefore required to generate new samples.

6.2 Functional (and other) validation(s)

We now present results about the functional validation of the proposed FPGA implementation of the LPPN processor, in nominal working conditions ($V_{DD} = 1.2$V, 25°C). To perform our validation, we have used a Sakura-G board. The LPPN processor is hosted on the main FPGA, namely a Xilinx Spartan-6 LX75, while the clock, inputs and PC interfacing are deployed within the control FPGA, a Xilinx Spartan-6 LX9. The clock frequency has been set at 13.56MHz, according to popular choices for RFID authentication tokens. In Figure 6, the probability of error (top) and value of the 6-bit $CNTL$ signal (bottom) in nominal condition versus the SAR algorithm's steps in the error control module are shown. The final probability of error is 0.28, which is within the bounds targeted in [KSD+20, KBS+18], while the value of the $CNTL$ signal is 34 in decimal representation. Clearly, the device is able to lock into secure operating conditions.

An evaluation of the proposed LPPN processor under different voltage and temperature working conditions is additionally reported in Appendix B. In short, it exhibits a good tolerance to changes of the temperature and a stronger dependency of the error probability in the supply voltage, justifying the previous fault detection scheme. A side-channel evaluation of the FPGA processor is also given in Appendix C. It estimates the side-channel Signal-to-Noise Ratio (SNR) of different target intermediate computations in the different (AND and XOR) stages of the LPPN processor [Man04]. Those results confirm the ASIC ones in [KBBS20] and show that the best targets are the AND stage and the last XOR stage, both of them exhibiting quite low SNRs though (in the 10^{-5} range).

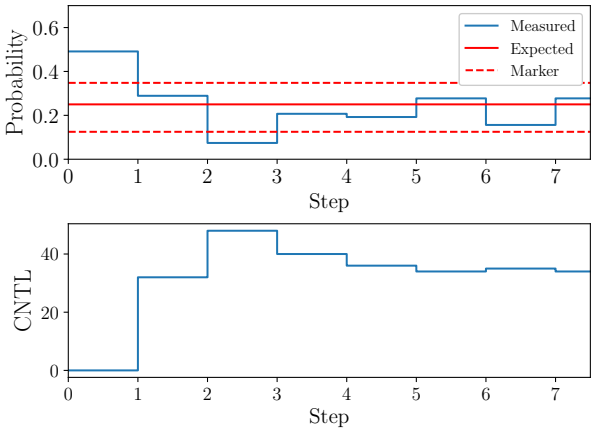


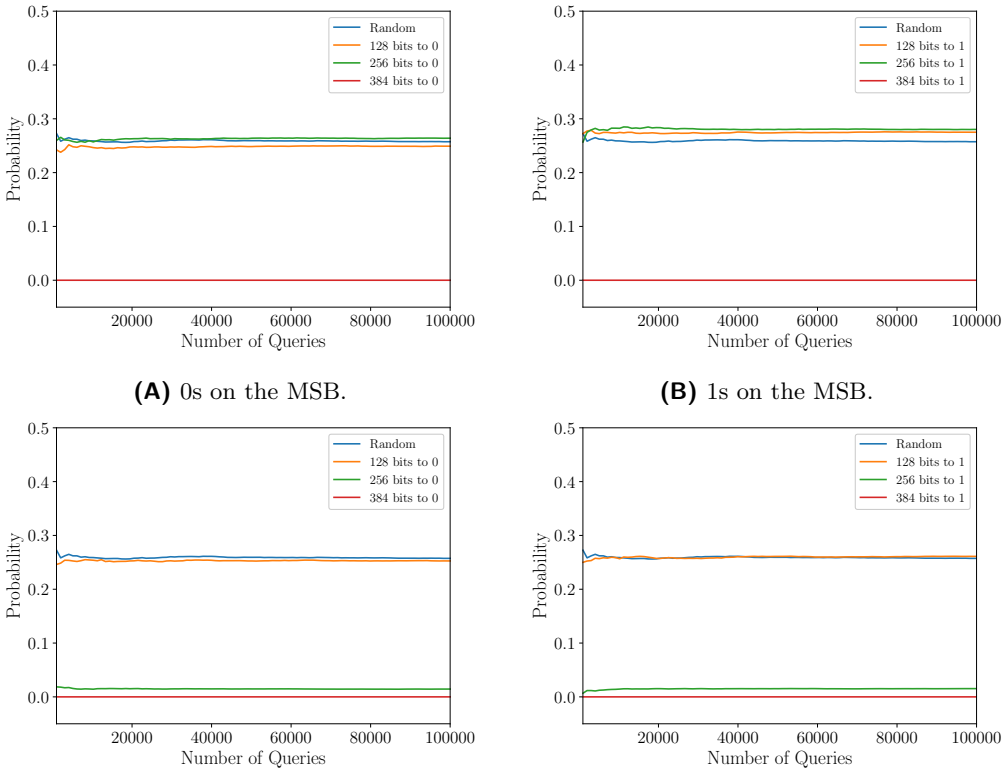
Figure 6: LPPN processor calibration: error probability (top) and control signal (bottom).

6.3 Data dependencies and mitigations

As discussed in the first sections of the paper, the probability of error of concrete ϵ-PIPs can suffer from data dependencies. So far, we did not discuss the input dependency issue (leveraging the observation that it is limited in the parallel then serial architectures we considered) and we showed a reduction proving that the LPN problem remains hard with output dependencies. In this section, we therefore study how much these dependencies can be observed in our FPGA prototype. We confirm that input dependencies are limited by design, highlight stronger output dependencies and discuss tracks to reduce them.

Input dependencies. To investigate the effect of input data dependency on the probability of error, we have performed filtering attacks on the FPGA implementation of the LPPN processor. In such attacks, the adversary isolates input challenges that she expects to lower the error probability. In particular, and as observed in [KSD+20, KBS+18], input challenges with low Hamming weights have a reduced error probability. So the main question for the security of an LPPN processor is the amount of filtering needed to significantly reduce the error probability. For this purpose, we analyzed this probability of error for input challenges with various Hamming weights, and also compared the situation when LSBs and MSBs are set to zero. The latter is motivated by the observation that the serial part of our implementation is asymmetric. Furthermore, we did not limit the investigation to forcing bits to zeros but also considered forcing them to ones. Concretely, we mimicked filtering attacks by setting the number of zeros or ones to 0, 128, 256 and 384. Experiments have been structured as follows. For each filtering attack, we have collected 10^5 bits that have been generated using a random fixed key. The LPPN processor has been calibrated at the beginning of each experiment with the same fixed key but with random challenges (we assume that the adversary has no control of the challenges, since the LPPN does not output anything during this phase), operating at 1.2V and room temperature.

In Figure 7, the input data dependency effect on the output probability of error is shown for the considered filtering attacks. Attacks exploiting the MSBs show that the probability of error collapse to 0 when the adversary forces 384 bits (to zero or one). Attacks exploiting the LSBs show that the probability of error already collapses to 0 when the adversary forces 256 bits (to zero or one). Even in that case, keeping one challenge out of 2^{256} is not a practical attack path against our 512-bit LPPN processor (since it is more than the security that the corresponding LPN problem guarantees). The fact that filtering the LSBs leads to stronger reductions of the error probability can be explained by



(A) 0s on the MSB.

(B) 1s on the MSB.

(C) 0s on the LSB.

(D) 1s on the LSB

Figure 7: Inputs' filtering attack on the proposed LPPN processor.

the asymmetry of our design. Precisely, MSB-based filtering attacks infer a certain value of the bits that experience longer logical paths in the serial stage, and therefore a 256-bit attack would force a 0 (or 1) on 3 XOR gates' input. On the other hand, LSB-based filtering attacks targeting 256 bits would force a 0 (or a 1) on 4 XOR gates' input. This difference is directly reflected in the capability of the serial XOR stage to produce glitch events, making the latter attack case more powerful. This observation is enforced by the linearity of the delay distribution over the serial XOR stages towards the P output with the depth of the logical cone, as remarked by the static timing analysis tool.

Output dependencies. We similarly investigated output dependencies of our LPPN samples' error probability by estimating this error probability in function of the correct output of the inner product computations. These estimated probabilities versus the number of queries are depicted in Figure 8. The normalized difference between the error probabilities of our baseline design after 10^5 bits is Δ=8.2%, indicating that this prototype implementation suffers from non negligible output dependencies. As a first step in order to mitigate this issue, we have designed an improved version of our design that makes use of dummy (data-independent) processing stages. That is, to reduce the normalized difference Δ, we have complemented the basic LPPN design in Figures 3A-3B with a dummy circuit that generates additional data-independent glitches. This dummy circuit computes the parity of a 128-bit string, adopting a 7-layer parallel XOR tree structure that has the same logical depth as the AND layer plus the parallel XOR stages in the inner product combinational network of Figure 3B. The output bit is XORed twice to the serial XOR stage of the LPPN design. More precisely, the output of the dummy circuit is XORed in

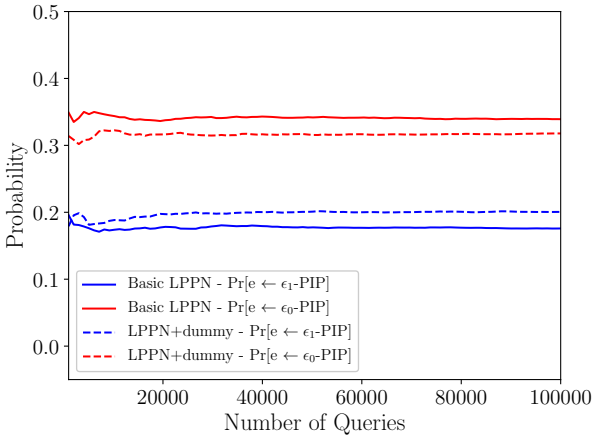


Figure 8: Output dependency on the proposed LPPN implementations.

the first and in the last stages of the serial XOR part, in order to increase its effectiveness in generating glitches while removing its functional value, as shown in Figure 9. Adopting a dummy circuit conceived this way can be justified as follows. First, designing a parity circuit with a logical cone that is similar to logical cones that produce the 8-bit vector used as input for the serial stage of the LPPN processor has good timing features. In fact, the output bit of the dummy circuitry arrives at the serial stage with a delay that is similar to the other bits entering in the same stage. Thus, no modification on the LPPN processor are required to ensure a correct timing of the circuit (regarding the VDL). Next, since those dummy bits remain unknown to the adversary and cannot be filtered, the glitches they cause consequently reduce the output data dependencies, as depicted in Figure 8. The normalized difference using a single dummy bit is already Δ=5.8%.

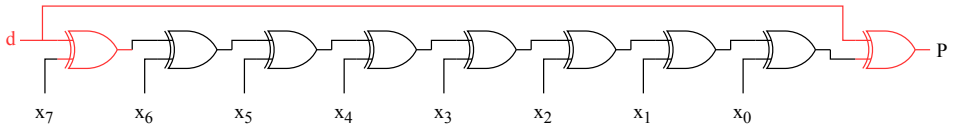


Figure 9: Modified implementation of the serial XOR stage with a dummy bit.

We assume more dummy bits could further reduce those dependencies, which we leave as an interesting scope for further research, together with other design tweaks that could be investigated such as the exploitation of dual-rail logic styles [TV04].

6.4 Putting things together

We conclude by evaluating the security that our LPPN processor provides, by plugging our estimated Δ's into the bounds of Section 4. Concretely, we consider three cases. The first two cases are the baseline FPGA design ($\Delta \approx 8.2\%$) and the one improved with dummy operations ($\Delta \approx 5.8\%$) of the previous subsection. The third case is a masked LPPN.

In this last case, we refer to the SNR values given in Appendix C which are in the 10^{-5} range.[8] Given that (*i*) the data complexity of a side-channel attack against a target intermediate computation is inversely proportional to the SNR, and (*ii*) the attack against the incorrect shared inner product that we target is an SPA, we can directly conclude that

[8] We computed this SNR based on the incorrect output value and obtained similar results.

the δ value of Section 5 will be close to $\frac{1}{2}$ in our case (e.g., it is lower than the success rate for $n = 1$ estimated in [KBBS20] which exhibits larger SNR values). As a result, the security of the (masked) LPPN samples will be close to the one of LPN samples.

As for the two first cases, assuming the LPPN processor provides LPN-OD samples with error parameters $0.25 \pm \Delta$ and $0.25 \mp \Delta$, Corollary 1 shows that retrieving the key from these samples is at least as hard as solving LPN with parameter ϵ where $\epsilon = \frac{0.25-\Delta}{1-2\Delta}$. We therefore base the security of LPN-OD$^{512}_{0.25\pm\Delta,0.25\mp\Delta}(\boldsymbol{k})$ on the complexity of the best known attacks on LPN$^{512}_{\frac{0.25-\Delta}{1-2\Delta}}(\boldsymbol{k})$. Indeed, breaking our instances with a lower complexity would imply better attacks on the well established LPN problem. Concretely, our LPPN processor relies on the security of LPN$^{512}_{0.217}(\boldsymbol{k})$ and on LPN$^{512}_{0.200}(\boldsymbol{k})$ for $\Delta = 8.2\%$. For such parameters, we estimate a security of 80 bits. According to the work [BTV16], LF2 is the best attack against LPN instances with $n = 512$ and $\epsilon \approx 0.25$. It gives a theoretical time complexity of 99 bits (for a query complexity of 88 bits). To find these numbers, we used the formulas and methods of [BTV16]. To conduct LF2, an adversary can fix two parameters a and b such that $a \cdot b > n$, where n is the size of our key (512). The best attacks were reached for $a = 6$ and $b = 86$. We take a safety margin over the theoretical security of 99 bits to cover possible gaps between theoretical and practical results for LF2.

7 Conclusions and further research

Our results suggest that inexact computing is a promising candidate for the secure and efficient implementation of LPN-based authentication protocols. They show that some of the defaults that inevitably occur when instantiating cryptographic primitives based on hard physical learning problems can be captured by formal reductions to known (mathematical) hard learning problems. These results naturally suggest the study of the Learning with Physical Errors (LWPE) problem outlined in [KSD+20] and its application to the secure and efficient implementation of PQ cryptography as a challenging next step. That is, can we generate samples following more complex distributions (e.g., as needed for LWE) with physical processes and can we reduce their imperfect implementations to known (mathematical) hard learning problems in this case as well? The evaluation and analysis of our proposed designs on other FPGAs is another interesting open problem.

Acknowledgments

François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in parts by the European Union through the ERC project SWORD (724725).

References

[AAB17] Benny Applebaum, Jonathan Avron, and Chris Brzuska. Arithmetic cryptography. *J. ACM*, 64(2):10:1–10:74, 2017.

[AG10] Sanjeev Arora and Rong Ge. Learning parities with structured noise. *Electron. Colloquium Comput. Complex.*, 17:66, 2010.

[AHM14] Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight authentication protocols on ultra-constrained rfids - myths and facts. In *RFIDSec*, volume 8651 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2014.

[AMS+11] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, François-Xavier Standaert, and Christian Wachsmann. A formalization of the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412. IEEE Computer Society, 2011.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 291–308. ACM, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng 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.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *EuroS&P*, pages 353–367. IEEE, 2018.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

[BL12] Daniel J. Bernstein and Tanja Lange. Never trust a bunny. In *RFIDSec*, volume 7739 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2012.

[BL17] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nat.*, 549(7671):188–194, 2017.

[BTV16] Sonia Bogos, Florian Tramer, and Serge Vaudenay. On solving LPN using BKW and variants. *Cryptography and Communications*, 8(3):331–369, 2016.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106. IEEE Computer Society, 2011.

[Cha14] Ken Chapman. *Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources*, 2014. Application Note: Spartan-6 Family, Virtex-6 Family, Series-7 FPGAs, Xilinx Inc.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552. Springer, 2010.

[DFH+16] Stefan Dziembowski, Sebastian Faust, Gottfried Herold, Anthony Journault, Daniel Masny, and François-Xavier Standaert. Towards sound fresh re-keying with hard (physical) learning problems. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 272–301. Springer, 2016.

[DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *J. Cryptol.*, 32(4):1263–1297, 2019.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 355–374. Springer, 2012.

[DV13] Alexandre Duc and Serge Vaudenay. HELEN: A public-key cryptosystem based on the LPN and the decisional minimal distance problems. In *AFRICACRYPT*, volume 7918 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2013.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GLS14] Lubos Gaspar, Gaëtan Leurent, and François-Xavier Standaert. Hardware implementation and side-channel analysis of lapin. In *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2014.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. ACM, 2008.

[GR10] Swaroop Ghosh and Kaushik Roy. Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proc. IEEE*, 98(10):1718–1751, 2010.

[GRS08a] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. Hb$^{\#}$: Increasing the security and efficiency of hb^{+}. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2008.

[GRS08b] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. How to encrypt with the LPN problem. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2008.

[Hås97] Johan Håstad. Some optimal inapproximability results. In *STOC*, pages 1–10. ACM, 1997.

[HB01] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on ring-lpn. In *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2012.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. *IACR Cryptol. ePrint Arch.*, 2020:1003, 2020.

[JW05] Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[KBBS20] Dina Kamel, Davide Bellizia, Olivier Bronchain, and François-Xavier Standaert. Side-channel analysis of a learning parity with physical noise processor. *J. Cryptogr. Eng.*, pages 1–9, 2020.

[KBS+18] Dina Kamel, Davide Bellizia, François-Xavier Standaert, Denis Flandre, and David Bol. Demonstrating an LPPN processor. In *ASHESCCS*, pages 18–23. ACM, 2018.

[Kea93] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. In *STOC*, pages 392–401. ACM, 1993.

[KPC+11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 7–26. Springer, 2011.

[KSD+20] Dina Kamel, François-Xavier Standaert, Alexandre Duc, Denis Flandre, and Francesco Berti. Learning with physical noise or errors. *IEEE Trans. Dependable Secur. Comput.*, 17(5):957–971, 2020.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.

[Man04] Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[Pie12] Krzysztof Pietrzak. Cryptography from learning parity with noise. In *SOFSEM*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012.

[PMB+16] Oto Petura, Ugo Mureddu, Nathalie Bochard, Viktor Fischer, and Lilian Bossuet. A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In Paolo Ienne, Walid A. Najjar, Jason Helge Anderson, Philip Brisk, and Walter Stechele, editors, *26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016*, pages 1–10. IEEE, 2016.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

[Reg10] Oded Regev. The learning with errors problem (invited survey). In *Computational Complexity Conference*, pages 191–204. IEEE Computer Society, 2010.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004.

[UHO15] Miho Ueno, Masanori Hashimoto, and Takao Onoye. Real-time on-chip supply voltage sensor and its application to trace-based timing error localization. In *21st IEEE International On-Line Testing Symposium, IOLTS 2015, Halkidiki, Greece, July 6-8, 2015*, pages 188–193. IEEE, 2015.

[YRG+18] Bohan Yang, Vladimir Rozic, Milos Grujic, Nele Mentens, and Ingrid Verbauwhede. ES-TRNG: A high-throughput, low-area true random number generator based on edge sampling. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):267–292, 2018.

[YS16] Yu Yu and John P. Steinberger. Pseudorandom functions in almost constant depth from low-noise LPN. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 154–183. Springer, 2016.

[YZW+19] Yu Yu, Jiang Zhang, Jian Weng, Chun Guo, and Xiangxue Li. Collision resistant hashing from sub-exponential learning parity with noise. In *ASIACRYPT (2)*, volume 11922 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2019.

[ZHL+13] L. Zhao, X. Hu, S. Liu, J. Wang, Q. Shen, H. Fan, and Q. An. The design of a 16-channel 15 ps tdc implemented in a 65 nm fpga. *IEEE Transactions on Nuclear Science*, 60(5):3532–3536, 2013.

A The ASIC LPPN prototype

The ASIC design of LPPN is similar to the one in [KBS+18]. It consists of three parts:

- A 512-bit (64-bit parallel then 8-bit serial) inner product architecture;
- A variable delay line that outputs a delayed version of the clock to sample the output of the inner product block during its glitchy period. It features digitally-controlled delay elements with shunt capacitors via NMOS switches;
- A controller that regulates the delay of the clock through a 6-bit control signal.

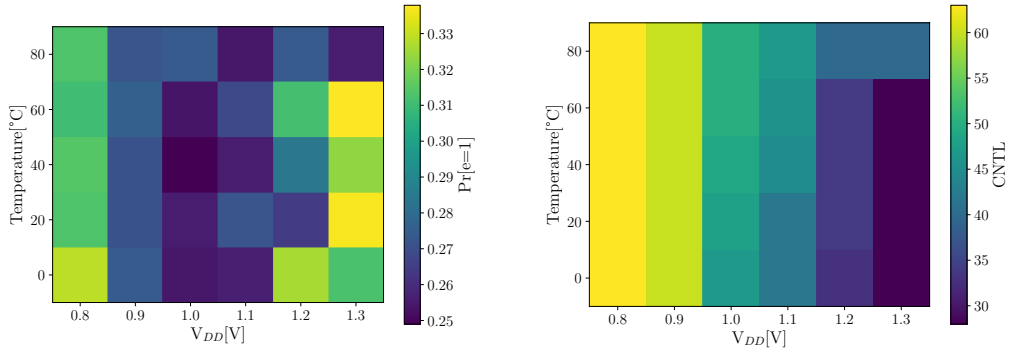
The operation is done in two steps. First, a calibration phase during which the error control is adjusted in order to maintain the required error probability ϵ during authentication. Second, the actual authentication. Concretely, the inner product block computes $\langle \boldsymbol{x}, \boldsymbol{k} \rangle$, where $\boldsymbol{k}$ is the secret key and $\boldsymbol{x}$ is the input, and generates a glitchy output parity bit (P). The variable delay line is designed such that it outputs a delayed clock that samples the inner product block output (P) while it is still in its glitchy period. During calibration only, the correct output $P_{correct}$ is also sampled such that the error controller computes the error signal e which is then counted via a 10-bit error counter over 1024 evaluations. To achieve the required error probability, we finally compare the error count to the target count (e.g., 256 for $\epsilon = 0.25$). Then, the 6-bit control signal is adjusted in a successive approximation scheme in order to set the delay of the variable delay line.

B Voltage and temperature validation

The outcome of the authentication phase for different voltage-temperature conditions is shown in Figure 10. It highlights that the LPPN processor is able to reach in-bound probabilities of error under the analyzed temperature variations, while more significantly suffering from voltage variations. The latter justifies the relevance of the sensor described in Section 6.1, which can detect voltage below 0.8V or above 1.3V.

C Side-channel security evaluation

We performed a preliminary evaluation of the side-channel leakage that our LPPN processor provides. To enable a simple interpretation of its outcome, we adopted Mangard's Signal-to-Noise Ratio (SNR) as a security metric [Man04]. The SNR provides an intuitive



(A) **(B)**

Figure 10: Probability of error (A) and value of the 6-bit *CNTL* signal (B) after the calibration phase, in function of the power supply voltage and working temperature.

quantification of the observed side-channel leakages and the complexity of a DPA targeting an intermediate computation with a given SNR is inversely proportional to this SNR.

The SCA evaluation of the proposed FPGA implementation has been carried out adopting a Picoscope 5244B as digital oscilloscope, running at 500MSa/s with 12-bit of vertical resolution. We have used a Tektronix CT-1 inductive probe to measure the current absorbed by the device during the LPPN computation post-calibration in nominal condition. A properly generated trigger signal ensured a good synchronization of the power traces. In total, we have collected $4{\times}10^6$ traces. The challenge and the secret have been changed randomly for each trace, in order to stimulate as much as possible all the logical paths inside the LPPN processor. We have conducted a thorough evaluation on each bit of the Inner Product module in Figure 3B. Maximum peaks of SNR for each bit are reported in Figure 11A and their average values are in Figure 11B. These figures show similar trends as for the ASIC prototype in [KBBS20]. Namely, the best SNR is observed for the initial AND gate, while digging through the inner product computations first reduces this SNR before increasing it. Overall, this trend can be explained by the combined effect of the algorithmic noise (which is maximum in the early stages) and the accumulated jitter (which is maximum in the last stages), leading to the worst results in the middle stage where both effect are combined in a noisy fashion.

Concretely, since exploiting the leakage of the AND gate is also the easiest option from an algorithmic viewpoint, as it enables simple divide-and-conquer attacks, we conclude that this is the sweet spot for an adversary. (Exploiting the leakage of the output bit would require mounting more complex analytical attacks). Yet, even for this sweet spot, the observed SNR values are still quite small (in the range of 5×10^{-5}) which would be easily combined with countermeasures like masking. For example, taking a factor 10 of security margin to cover multivariate attacks, we would remain with leakage metrics significantly below 10^{-3}, so that a few shares would make the complexity of a side-channel attacks against such an LPPN prototype prohibitive [DFS19]. Combined with the key-homomorphic feature of the inner product computations which enable masking with linear time and randomness overheads, we can conclude that the proposed LPPN prototype is an interesting candidate for cost-effective countermeasures against side-channel attacks.

D Implementation figures and discussion

In this appendix, we finally discuss the area requirements and performances of our LPPN prototype on Xilinx Spartan-6 FPGAs. We then briefly discuss its pros and cons compared

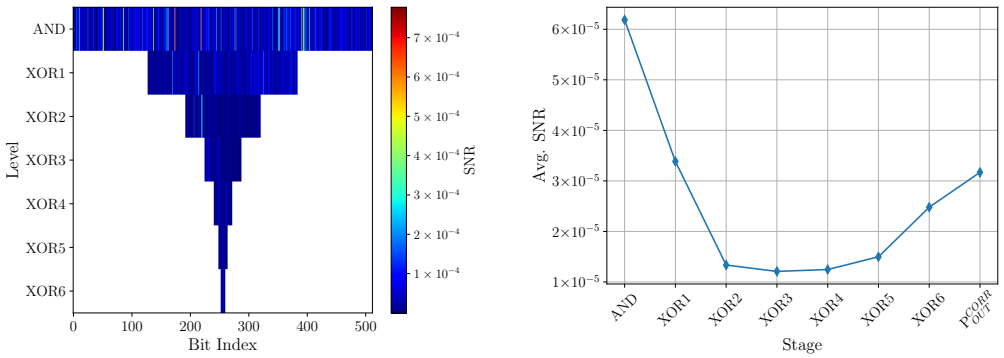


Figure 11: Maximum peaks of SNR by bit of the inner product module (AND and parallel XOR) (a), and average among peak values of the whole inner product module by layer (b).

to a classical LPN-based implementation leveraging a RNG. A detailed summary of the area requirement of our design is reported in Table 1. It is straightforward to notice that the Inner Product submodule is dominant on the overall cost, both in terms of LUTs and flip-flops, accounting for respectively 79.4% and 94.6% of the whole LPPN processor. Since the errors of the LPPN prototype are generated physically (by means of glitching events and occurrence of metastability), the Inner Product has not been optimized in terms of LUTs occupation and therefore all the AND and XOR gates discussed in Section 6 have been implemented as 2-input LUTs. The VDL has a very small impact on the overall design (7.1% of combinational resources), being optimized to be compact and leveraging the CARRY4 and MUXF7/8 units to obtain a fine-grain delay. The Error Control has a negligible impact on the overall utilization too, as its design requires only few LUTs and registers. The Voltage Sensor requires 8.1% of the overall LUT count, hence is comparable to the VDL in terms of cost, and it is able to provide resilience against post-calibration alteration of the working conditions. (It may not be needed in a threat model where the adversary cannot alter the power voltage supply of the LPPN processor). The maximum operating frequency of the LPPN prototype is 66.65MHz. It is imposed by the Inner Product architecture and leads to a throughput at maximum frequency of 1.04Mbps.

Table 1: Resource utilization of the LPPN processor on the Xilinx Spartan-6 FPGA.

Module	LUTs	Regs
LPPN	1290 (100%)	1086 (100%)
Inner Product	1024 (79.4%)	1027 (94.6%)
VDL	92 (7.1%)	-
Error Control	50 (3.9%)	39 (3.6%)
Voltage Senser	104 (8.1%)	8 (0.7%)
Others	20 (1.5%)	1074 (1.1%)

Comparing the LPPN assumption with a classical LPN implementation relying on a RNG is not straightforward. The design philosophy behind LPPN aims at removing the need of a RNG from the construction, embedding the generation of errors by means of inexact inner product computations. So it would require comparing the cost of an explicit RNG, the design of which is in itself a non-trivial design challenge: it usually requires good design skills and understanding of the physics behind the circuits that harvest and

Table 2: Summary of RNGs implementations on Xilinx Spartan-6 from [PMB+16].

RNG Type	Area (LUTs/Regs)	Special Macros	Bit Rate [Mbps]	Feasibiliy & Repeatability
ERO	46/19	No	0.0042	Very Good
COSO	18/3	No	0.54	Very Poor
MURO	521/131	No	2.57	Good
PLL	34/12	Yes	0.44	Medium
TERO	39/12	No	0.625	Very Poor
STR	346/256	No	154	Poor

digitize entropy to generate random bits.[9] Especially on FPGAs, designing RNGs is a delicate task, due to the limited degrees of freedom they offer. Many aspects have to be considered for such designs, including area cost and throughput but also feasibility and reliability (e.g., in terms of possibilities to move the macro inside the FPGA and still have it functional). In [PMB+16], an interesting survey of different FPGA-suitable RNG architectures is reported, tailoring three commercial FPGAs as a case study. In Table 2, we report a summary of RNG implementation results from this reference, focusing on the Xilinx Spartan-6 FPGA case. It has to be noted that achieving at the same time high throughput, small area footprint and feasibility/reliability is quite hard. Besides, the error generation is a security-critical ingredient for LPN-based implementations, and therefore it requires robustness against different working conditions. So while the investigation of LPPN prototypes is still in an early stage, we use these results as an indication that the way these prototypes generate erroneous outcomes with reasonable security guarantees against environmental changes may be competitive with standard LPN implementations, in particular in situations where side-channel security is a concern.

[9] We do not consider PRNGs as an option for the generation of the LPN errors since it would require side-channel countermeasures to reach the same physical security guarantees as our LPPN prototype provides. By contrast, using a PRNG is a natural option for generating the (public) challenges.