Analyzing the Shuffling Side-Channel Countermeasure for Lattice-Based Signatures

Peter Pessl

IAIK, Graz University of Technology, Graz, Austria
peter.pessl@iaik.tugraz.at

Abstract. Implementation security for lattice-based cryptography is still a vastly unexplored field. At CHES 2016, the very first side-channel attack on a lattice-based signature scheme was presented. Later, shuffling was proposed as an inexpensive means to protect the Gaussian sampling component against such attacks. However, the concrete effectiveness of this countermeasure has never been evaluated.
We change that by presenting an in-depth analysis of the shuffling countermeasure. Our analysis consists of two main parts. First, we perform a side-channel attack on a Gaussian sampler implementation. We combine templates with a recovery of data-dependent branches, which are inherent to samplers. We show that an adversary can realistically recover some samples with very high confidence.
Second, we present a new attack against the shuffling countermeasure in context of Gaussian sampling and lattice-based signatures. We do not attack the shuffling algorithm as such, but exploit differing distributions of certain variables. We give a broad analysis of our attack by considering multiple modeled SCA adversaries.
We show that a simple version of shuffling is not an effective countermeasure. With our attack, a profiled SCA adversary can recover the key by observing only 7 000 signatures. A second version of this countermeasure, which uses Gaussian convolution in conjunction with shuffling twice, can increase side-channel security and the number of required signatures significantly. Here, roughly 285 000 observations are needed for a successful attack. Yet, this number is still practical.
Keywords: Lattice-Based Cryptography, BLISS, Side-Channel Analysis, Countermeasures

1 Introduction

Quantum computers are a serious threat to a majority of currently in-use public-key cryptosystems. Although powerful enough quantum computers might not be available in the near future, their possible advent causes concerns and has already led to official recommendations from government bodies, such as the NSA [16,22]. Furthermore, standardization agencies are starting to investigate post-quantum alternatives [6]. Very recently, also Google began experimenting with post-quantum key-exchange algorithms in their Chrome browser [1,3].

Lattice-based cryptography is a very promising candidate for the post-quantum world. It proved to be very versatile and offers practical realizations of many public-key building blocks. When it comes to digital signatures, the Bimodal Lattice Signature Scheme (BLISS), which was presented by Ducas, Durmus, Lepoint, and Lyubashevsky [7] at CRYPTO 2013, is an attractive option. This is due to its efficiency both in terms of runtime and parameter sizes. Signature and public key sizes are in the range of current RSA moduli, which is a significant improvement over many earlier proposals.

There already exists a large body of work targeting efficient implementation of lattice-based primitives. Even when only considering BLISS, these range from hardware implementations [18] to microcontrollers [17,19] and PCs [23]. However, up until very recently the implementation-security aspect was pretty much neglected. The first side-channel attack on a lattice-based signature scheme, namely BLISS, was presented at CHES 2016 by Groot Bruinderink et al. [10]. They use a cache attack to recover some of the outputs of a Gaussian sampling algorithm. By combining information from multiple signatures and respective identified samples, they are able to recover the key. Note that Gaussian samplers play an integral part in most lattice-based schemes and their implementations. Hence, this type of attack might be applicable to a multitude of settings.

Shuffling was proposed by Saarinen [21] as a countermeasure against such an attack. Instead of securing the sampler itself, which would come at a hefty price, one could simply generate n Gaussian samples using an unprotected implementation and then randomly permute them. Shuffling is easy to implement and has a relatively low runtime overhead. This makes it especially attractive for use in low-resource devices, such as microcontrollers. However, the concrete security gains achieved by shuffling have thus far never been analyzed. As a consequence, convincing security arguments are still sorely lacking.

Our Contribution. In this paper, we tackle the above problem and present an in-depth analysis of shuffling in context of lattice-based signatures. Our analysis consists of two main parts, a side-channel analysis and a new attack on shuffling.

In the first part, we perform a side-channel attack on a Gaussian sampler implementation running on an ARM microcontroller. Our attack combines two methods. First, we recover the control flow of the sampling procedure. As samplers, including the one used by us, require data-dependent branches and are not inherently constant runtime, this already allows to narrow down the possible samples. And second, we use templates to uniquely identify the sampled value. While this attack is not able to identify all samples, it can recover certain values with very high confidence.

In the second part of our shuffling analysis, we present a new attack on the countermeasure. We perform an *un-shuffling*, i.e., reassign some recovered samples to the corresponding part of the signature output. After having collected enough matching pairs over multiple signatures, we can recover the private signing key. We stress that we do not attack the shuffling algorithm as such, we do not even consider its leakage in our analysis. Instead, we exploit the difference in distributions of Gaussian samples (high standard deviation) and a specific key-dependent intermediate (low standard deviation).

As we aim for a broad analysis, we evaluate this attack given several modeled side-channel adversaries. They are largely based on the previous side-channel analysis, but to test the theoretical boundaries of the countermeasure we also include an ideal attacker who is able to recover all samples. We also consider two different versions of the shuffling countermeasure. Our analysis shows that the simpler variant does not provide a noteworthy increase in side-channel security. Our ideal attacker succeeds using only 40 signatures. With 7 000 signatures, the modeled adversary which is closest to our side-channel analysis can also easily recover the key. However, the second shuffling version, which uses Gaussian convolution and shuffles twice, can increase the number of observed signatures required for an attack significantly. Yet, with around 260 000 signatures (for both mentioned adversaries) an attack is still practical and possible[1].

Finally, note that while we focus on BLISS, Gaussian sampling is required for most lattice-based schemes. Thus, the shuffling countermeasure and also our attack could be used for a wide range of implementations.

Outline. In Section 2, we recall BLISS, discrete Gaussians and proposed samplers. Then, in Section 3 we discuss previous work on SCA and countermeasures on BLISS. We evaluate the side-channel leakage of a concrete Gaussian sampler implementation in Section 4. Using the results of this side-channel analysis, we present an attack on the shuffling countermeasure and also discuss its outcome in Section 5. Finally, we conclude in Section 6.

2 BLISS and Gaussian Samplers

We now give a brief description of BLISS [7]. We then go on and describe the discrete Gaussian distribution and methods to sample from it.

2.1 BLISS - Bimodal Lattice Signatures

The most efficient instantiation of BLISS works with polynomials over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. We will later use the fact that the multiplication of two polynomials $\mathbf{a}, \mathbf{b} \in \mathcal{R}_q$ can be written as a matrix-vector product, i.e., $\mathbf{ab} = \mathbf{aB} = \mathbf{bA}$, where the columns of matrices $\mathbf{A}, \mathbf{B}$ are negacyclic rotations of $\mathbf{a}$ and $\mathbf{b}$, respectively.

Key generation and signature verification do not play a role in our later analysis, here we refer to [7]. Signature generation is described in Algorithm 1. It takes as input a message μ, a public key $\mathbf{A}$, and a private key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)$, with $\mathbf{s}_1$ a polynomial with exactly $\delta_1 n$ coefficients in $\{\pm 1\}$, $\delta_2 n$ coefficients in $\{\pm 2\}$, and all other elements being 0. First, two noise polynomials $\mathbf{y}_1, \mathbf{y}_2$ are sampled from a discrete Gaussian distribution D_σ. The intermediate $\mathbf{u}$ is hashed together with the message, where H outputs a bit vector $\mathbf{c}$ of length n and (small) hamming

[1] These numbers assume recoverability of bit b for each signature (cf. Section 5.3).

Algorithm 1. BLISS Signature Algorithm

```
Input: Message $\mu$, public key $\mathbf{A} = (\mathbf{a}_1, q-2)$, private key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)$
Output: A signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
1: $\mathbf{y}_1 \leftarrow D_\sigma^n$, $\mathbf{y}_2 \leftarrow D_\sigma^n$
2: $\mathbf{u} = \zeta \cdot \mathbf{a}_1\mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$
3: $\mathbf{c} = \mathrm{H}(\lfloor \mathbf{u} \rceil_d \bmod p || \mu)$
4: Sample a uniformly random bit $b$
5: $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1\mathbf{c}$
6: $\mathbf{z}_2 = \mathbf{y}_2 + (-1)^b \mathbf{s}_2\mathbf{c}$
7: Continue with some probability $f(\mathbf{Sc}, \mathbf{z})$, restart otherwise (see [7])
8: $\mathbf{z}_2^\dagger = (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d)$
9: return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
```

n	q	σ	δ_1, δ_2	κ	d
512	12289	215.73	0.3, 0	23	10

Table 1. BLISS-I parameter set

weight κ. The noise polynomials are then added to $\mathbf{s}_1\mathbf{c}$ and $\mathbf{s}_2\mathbf{c}$, respectively. A subsequent rejection-sampling step prevents leakage of the key. Finally, the compressed signature is returned[2]. Throughout this paper, we use the BLISS-I parameter set [7] given in Table 1. It provides a security level of 128 bit.

2.2 Discrete Gaussians

We denote with D_σ the discrete Gaussian distribution with standard deviation σ and zero mean; we use $y \leftarrow D_\sigma$ for variables sampled from this distribution. The probability-mass function $D_\sigma(x) = \rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$, with $\rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$ and the normalization constant $\rho_\sigma(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_\sigma(k)$. With D_σ^n, we denote the n-dimensional extension. Samples from D_σ^n can simply be generated by independently sampling n times from D_σ.

Implementation of Gaussian samplers. The emergence of lattice-based cryptography and its reliance on discrete Gaussian noise led to a large number of proposed sampler architectures. Apart from generic methods like rejection sampling and inversion sampling, these also include, e.g., the Knuth-Yao random walk [8], the Ziggurat method [4], and arithmetic coding [21].

Compared to lattice-based public-key encryption [14], the standard deviation required for BLISS is relatively high. This makes samplers requiring large precomputed tables less attractive, especially for constrained devices and their usually low storage capacities. For this reason, Pöppelmann et al. [18] proposed an optimized sampler which is based on the inversion method. Since their approach is tailored for low-resource devices and also an ideal candidate for use

[2] The constants ζ, d, p are used for compression purposes. For details, see [7].

with the shuffling countermeasure, we use their algorithm in our work and now give a more detailed description.

For inversion sampling, one first precomputes a cumulative distribution table (CDT), i.e., a table $T[y] = \mathrm{P}(x < y | x \leftarrow D_\sigma^+)$ for $y \in [0, \tau\sigma]$. Here, τ denotes the tail-cut factor which is required due to the infinite support of D_σ. Thanks to symmetry of D_σ, sampling can be easily reduced to sampling from the one-sided distribution D_σ^+ with support $[0, \tau\sigma]$ and then sampling a random sign bit. As the statistical distance to a true discrete Gaussian must be kept low, the entries of T need to be stored with a very high precision, e.g., 128 bit.

For actual sampling, one generates a uniformly random $r \in [0, 1)$ and returns the y satisfying $T[y] \leq r < T[y+1]$ (using a binary search in T). To reduce the table size and speed up sampling, Pöppelmann et al. propose the following optimizations. They save memory by using Gaussian convolution. They set $k = 11$, $\sigma' = \sigma/\sqrt{1+k^2} \approx 19.53$ and sample two values $y', y'' \leftarrow D_{\sigma'}$. They then combine them to $y \leftarrow D_\sigma$ by setting $y = ky' + y''$. Furthermore, they speed up sampling by using a byte-oriented guide table I. Each entry $I[r_0]$ stores the smallest interval $(\min_{r_0}, \max_{r_0})$ with $T[\min_{r_0}] \leq r_0/256$ and $T[\max_{r_0}] \geq (r_0+1)/256$. By using this table, the range for the following binary search can be immediately reduced to the interval $[\min_{r_0}, \max_{r_0})$.

The detailed sampling procedure is given in Algorithm 2. It uses a bytewise approach, where $T_j[i]$ denotes the j-th byte of $T[i]$. To save memory, the table T is stored in floating-point representation, using a mantissa table M and an exponent table E. For efficiency reasons Pöppelmann et al. actually store $T[y] = \mathrm{P}(x \geq y | x \leftarrow D_\sigma^+)$, i.e., $T[0] = 1$ and $T[y] > T[y+1]$. This is accounted for in the binary-search part. For further explanations we refer to [18].

Algorithm 2. CDT Sampler using Guide Tables [18]

```
Input: Guide table $I$, mantissa table $M$, exponent table $E$
Output: A value $y'$ sampled according to $D_{\sigma'}$
 1: Sample a uniformly random byte $r_0$
 2: $[\min, \max] = I[r_0]$
 3: $i = (\min + \max)/2$, $j = 0$, $k = 0$
 4: while max-min $> 1$ do
 5:     $t = T_j[i]$, with $T_j[i] = M_{j-E[i]}[i]$ or 0
 6:     if $t > r_j$ then
 7:         $\min = i$, $i = (i + \max)/2$, $j = 0$
 8:     else if $t < r_j$ then
 9:         $\max = i$, $i = (\min + i)/2$, $j = 0$
10:     else
11:         $j = j + 1$
12:         if $k < j$ then
13:             Sample uniformly random byte $r_j$, $k = j$
14: Sample a uniformly random bit $s$
15: if $s$ then return $-i$
16: else return $i$
```

3 Side-Channel Attacks and Countermeasures for Gaussian Sampling

When analyzing the components of BLISS for side-channel weaknesses, the Gaussian sampler appears to be a critical and especially hard to protect part. To the best of our knowledge, none of the samplers given in the previous section inherently feature a constant runtime or a complete absence of data-dependent branches. Thus, it should not come as a huge surprise that the first reported side-channel attack on lattice-based signatures, which we will now discuss, targets samplers. We will then also state possible countermeasures, including shuffling.

3.1 A Cache Attack on BLISS

At CHES 2016, Groot Bruinderink et al. [10] presented the first side-channel attack on BLISS. They perform a cache attack, i.e., observe time differences caused by the CPU cache, to partially recover the Gaussian noise vector $\mathbf{y}_1$. They analyze the susceptibility of two sampler implementations to such attacks in depth[3].

Their attack proceeds as follows. First, they need to observe the creation of multiple signatures $(\mathbf{z}_j, \mathbf{c}_j)$, where $\mathbf{z}_j$ refers to only the first signature polynomial $\mathbf{z}_1$ of the j-th signature. They then focus on line 5 of Algorithm 1, i.e., $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$. For each recovered Gaussian sample, an attacker can create an equation of form:

$$z_{ji} = y_{ji} + (-1)^{b_j} \langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle \tag{1}$$

Here, i denotes the index of the recovered Gaussian sample in the signature. z_{ji} and y_{ji} are the i-th coefficients of $\mathbf{z}_1$ and $\mathbf{y}_1$ in the j-th signature. $\mathbf{c}_{ji}$ denotes the i-th column of $\mathbf{C}_j$, which is the matrix used in the matrix-vector representation of polynomial multiplication.

The cache attack does not reveal the random but secret bit b. Therefore, Groot Bruinderink et al. keep only those equations which satisfy $z_{ji} = y_{ji}$, i.e., where $\langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle = 0$ and thus the value of b is irrelevant. They then build a matrix $\mathbf{L}$ where the columns are the filtered $\mathbf{c}_{ji}$. This matrix satisfies $\mathbf{s_1 L} = 0$. The key $\mathbf{s}_1$ can then be found in the kernel-space of $\mathbf{L}$. $\mathbf{s}_2$ can be reconstructed by using the relation between public and private key.

Groot Bruinderink et al. also consider a scenario where the information on the samples is not exact, but instead a small error is possible. Here, they formulate a lattice problem and use lattice-reduction techniques for key recovery.

3.2 Countermeasures

Protecting samplers from attacks like the one above seems to be difficult. While there exist methods for constant-runtime and protected sampling, they come at a hefty performance impact (see, e.g., [2]). Also, there do exist alternatives

[3] Further sampler architectures are discussed in the full version of [10].

to using (high-precision) Gaussian noise. However, they either do not apply to signature schemes [1] or they are suboptimal in terms of security or signature size [11,7].

Instead of protecting the sampler itself, one could also simply use an unprotected (or somewhat protected) sampler implementation to generate n samples and then randomly permute them. This breaks the connection between time of sampling and index in the signature and thus makes attacks more difficult. This shuffling countermeasure was first proposed by Roy et al. [20], albeit in the context of lattice-based public-key encryption. Recently, Saarinen [21] proposed a variant that uses shuffling multiple times (in conjunction with Gaussian convolution) for use in BLISS. For m stages, he sets $\sigma' = \sigma/\sqrt{m}$, then samples $\mathbf{y}^i \leftarrow D^n_{\sigma'}$ for $i = 1..m$ and computes $\mathbf{y} = \sum_i \text{Shuffle}(\mathbf{y}^i)$. However, neither Roy nor Saarinen provided an analysis of this countermeasure. Thus, its true effectiveness has still been unknown.

In this work, we will investigate two versions of shuffling in terms of security. First, we have a look at simple shuffling in combination with the sampler of Pöppelmann et al. [18]. And second, we will analyze an instantiation of multi-stage shuffling that was concretely proposed by Saarinen [21]. He suggests to combine two stages of shuffling with the Gaussian-convolution parameters of Pöppelmann et al.. Below we give a description of both versions.

Single-Stage Shuffling: $\mathbf{y}', \mathbf{y}'' \leftarrow D^n_{\sigma'}$, $\mathbf{y} = \text{Shuffle}(k\mathbf{y}' + \mathbf{y}'')$
Two-Stage Shuffling: $\mathbf{y}', \mathbf{y}'' \leftarrow D^n_{\sigma'}$, $\mathbf{y} = k \cdot \text{Shuffle}(\mathbf{y}') + \text{Shuffle}(\mathbf{y}\text{''})$

4 A Side-Channel Attack on a Gaussian Sampler

Before evaluating the shuffling countermeasure, it is important to understand how much information on Gaussian samples a side-channel attacker can realistically expect. For this reason, we now present a side-channel analysis of a sampler implementation. Recall that Gaussian sampling is a random process that does not involve any keying material. Also, its output is typically used only once. Hence, we are limited to single-trace SPA-style attacks.

4.1 Implementation and Measurement Setup

For our experiments, we implemented the Gaussian sampling procedure proposed by Pöppelmann et al. [18] in software. The contents of all required lookup-tables are directly taken from their open-sourced BLISS FPGA implementation. Note that our analysis focuses solely on sampling from $D_{\sigma'}$ (Algorithm 2), i.e., we do not use any leakage stemming from the Gaussian convolution step.

As a target platform, we chose a Texas Instruments MSP432 (ARM Cortex-M4F) microcontroller on a MSP432P401R LaunchPad development board[4]. For

[4] The design files of this development board are available online [24].

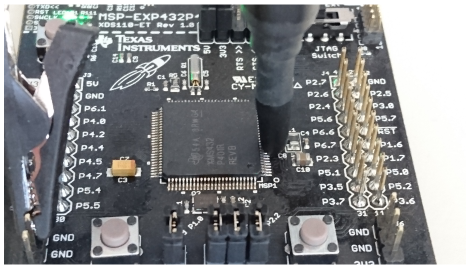


Fig. 1. Measurement setup. The EM probe is placed directly to the left of the external core-voltage regulation circuitry.

pseudo-random number generation we used the on-chip hardware AES accelerator in counter mode. While this setup is likely susceptible to DPA attacks [12,15], we do not use any leakage of the AES execution.

In our attack we exploit the EM side channel. As shown in Figure 1, we placed a Langer RF-B 3-2 near-field probe in proximity to the external core-voltage regulation circuitry. Note that for this setup, no spatial profiling of on-chip EM leakage is required. Also, we expect the results of power measurements to be somewhat similar. For our evaluation, we use a dedicated trigger that signals the start of a sampling procedure. Real-world attackers do not have this option and need to detect the 1024 calls to Algorithm 2 required for sampling $\mathbf{y}_1$. Such adversaries can use, e.g., trace alignment in combination with the methods described in the next section.

4.2 Reconstructing the Control Flow

When analyzing Algorithm 2, it becomes obvious that the data-dependent branches offer a lot of information on the sampled value. In fact, the return value can be uniquely determined by the first random byte r_0 and the control flow.

We recover the control flow using a trace-matching approach. For each possible conditional jump, we record a reference trace by computing the mean of multiple profiling traces at select points in time (in some cases just a single point) near the first occurrence of this branch. During the attack, we then compare these references to the attack trace by computing the mean of squared differences. Figure 2 illustrates that for some branches, the most information lies within a time shift of subsequent operations. In these cases, we use a single reference and match them at both locations. We then use the case with the lowest score. We repeat this matching process until the algorithm exits. The position of the respective next matching process is determined on basis of the previously taken branches. The final branch detection then also reveals the sign of the sampled value.

With the described method, we can reconstruct the control flow with perfect accuracy. This should not be surprising, when, e.g., observing the huge trace differences illustrated in Figure 2.

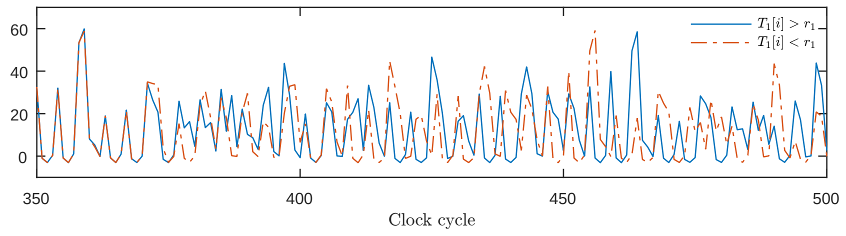


Fig. 2. Demonstration of a timing difference stemming from a branch inside the first loop iteration. After around cycle 420, the trace for $T_1[i] > r_1$ (blue, solid) trails by 8 clock cycles.

Note that, while we use device profiling for deriving the reference traces, there exist non-profiled alternatives. An attacker could, e.g., build the references on the fly after a visual inspection of a limited number of traces. Alternatively, he could use a clustering approach for determining the branches.

4.3 Determining the Sampled Values via Templates

In order to uniquely determine the sampled value, we recover the value of r_0 using a template attack [5]. For each possible control flow (up to a certain depth), we built templates for each value of r_0 that can potentially result in this flow. The points-of-interest for the attack were determined using a t-test, as proposed by Gierlichs et al. [9]. We limited the maximum number of used points to 8.

The outcome of the template attack is depicted in Figure 3. There we show a histogram of the maximum classification probabilities. In our implementation, the guide-table lookup already yields the final sample for 206 values of r_0. As seen in Figure 3a, we cannot determine the correct samples with high confidence in these cases. As our later analysis on the shuffling countermeasure requires such a high confidence, we have to discard these samples. This situation changes in cases that require a single comparison step in the binary-search algorithm, Figure 3b shows that 6.5 % of these samples can be determined with probability close to 1.

If more than a single comparison is required, then the template attack can recover the sampled value almost perfectly. The overall success rate here is 99.5 %. If we discard the 1 % of samples whose probability is lower than 0.90, then the success rate reaches 99.9 %.

5 An Analysis of the Shuffling Countermeasure

In this section, we give an in-depth analysis of the shuffling countermeasure. First, we give a brief discussion on its cost. Afterwards, we present an attack that can circumvent this countermeasure, albeit at the cost of requiring a higher number of recorded signatures. We state the performance of this attack with

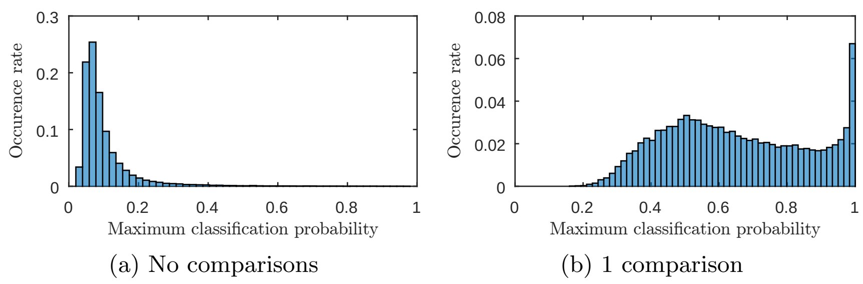


Fig. 3. Results of the template attacks for no or 1 comparison

regards to several modeled side-channel adversaries and variations of the countermeasure.

5.1 Cost

We evaluated the cost of shuffling by implementing the Fisher-Yates shuffling algorithm [13]. When run at 48 MHz, which is the maximum for our MSP432 evaluation platform, shuffling a vector of $n = 512$ entries took 1.5 ms. For comparison, sampling an element from $D^n_{\sigma'}$, which requires 512 calls to Algorithm 2, needs about 2.5 ms. For creating a single signature, 4 elements of $D^n_{\sigma'}$ need to be sampled. The shuffling operation is called either 2 or 4 times, depending on whether single-stage or two-stage shuffling is used. In the latter case, the total runtime of sampling is increased by 57 %, which is still relatively little when it comes to SCA countermeasures.

5.2 Considered Attackers

In order to allow a broad analysis of the shuffling countermeasure and to achieve easier reproducibility, we do not directly use the outcome of the attack described in Section 4. Instead, we use the results as a basis to model three side-channel adversaries. Each one is based on a different assumption on his capabilities. Note that all following descriptions are in context of sampling from the "small" $D_{\sigma'}$ and thus Algorithm 2, which is called 2048 times during signature generation. We do not use any leakage from the multiplication with k, the addition for Gaussian convolution, and even the shuffling algorithm itself. We do so to keep the analysis as generic and implementation-independent as possible.

A1 - perfect SCA adversary. This attacker is able to recover all generated samples. We use this adversary to evaluate the theoretical limits of the shuffling countermeasure.

A2 - profiled SCA adversary. This attacker is able to profile the device and perform a template attack. We assume that the attacker can correctly determine the entire control flow and is able to correctly classify all samples which required at least 2 comparisons in the binary-search step. For the analysis, we make a further simplification and only use samples with absolute above a certain threshold. This threshold is set so that all samples larger than it require at least 2 comparisons. All samples at and below the threshold are considered to be unknown.

A3 - non-profiled SCA adversary. This attacker is not able to profile and thus cannot perform a template attack. However, he is still able to reconstruct the control flow. All samples which are not uniquely determined by the control flow are considered to be unknown.

Adversary A2 is closest to the side-channel analysis given in the previous section. However, in this model we do not use any potentially classified samples which used only a single comparison (cf. Figure 3b) or the small portion of samples requiring 2 comparisons but being below the threshold. In return, we also ignore the very small error probability and assume that all reconstructed samples are correct.

For our particular BLISS parameter set and sampler implementation, we have the following concrete implications. For A2, the above mentioned threshold is 47, i.e., we say that the adversary can correctly classify all samples with absolute value larger than 47. Approximately 1.5 % of the samples from $D_{\sigma'}$ meet this restriction. The adversary A3 can correctly classify all samples with absolute value larger than 54, which amounts to only 0.53 % of all samples.

For each modeled adversary, we also evaluate two sub-scenarios with regards to the secret bit b used for computing $(-1)^b \mathbf{s}_1 \mathbf{c}$. First, we consider the case that the adversary can recover this bit with side-channel measurements using, e.g., methods akin to Section 4.2. And second, we also evaluate the case that this bit is unknown.

5.3 Attack without Shuffling

For key recovery, we use the relation also exploited by Groot Bruinderink et al. (cf. Section 3.1). We gather equations of the form $z_{ji} = y_{ji} + (-1)^{b_j} \langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle$ and then solve the resulting linear system. We do not consider error correction and require that these equations are correct.

If the entire $\mathbf{y}_1$ is known, which is the case for adversary A1, and no shuffling is used, then key recovery is trivial and requires only a single signature. $\mathbf{s}_1$ (or $-\mathbf{s}_1$) can be computed by solving the linear system given as $\mathbf{z}_1 - \mathbf{y}_1 = (-1)^b \mathbf{s}_1 \mathbf{c}$ for any value of b. Attackers A2 and A3 require multiple signatures in order to recover the key, even in the non-shuffled scenario. As our sampling procedure combines two samples $y', y'' \leftarrow D_{\sigma'}$ to $y = ky' + y''$, we can only recover samples y where the side-channel information reveals both y' and y''. Hence, A2 can recover a portion of $0.015^2 \approx 2.2 \cdot 10^{-4}$ of all samples, whereas for A3 this quantity decreases to $2.2 \cdot 10^{-5}$.

If the b_j are recoverable by using side-channel information, then we can combine n equations $z_{ji} = y_{ji} + (-1)^{b_j} \langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle$ into a linear system which can then simply be solved for the key $\mathbf{s}_1$. The expected number of signatures required to

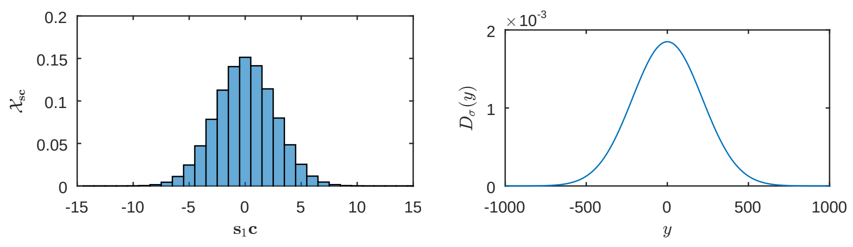


Fig. 4. Comparison of the coefficient-wise distribution of $\mathbf{s}_1\mathbf{c}$ ($\mathcal{X}_{\mathbf{sc}}$) and $\mathbf{y}$ (D_σ^n)

gather $n = 512$ classified samples and corresponding signature values is roughly 4 400 for A2 and 36 000 for A3. Note that in this non-shuffled scenario, the differences between A2 and the SCA from Section 4, i.e., not using all classifiable samples, have a significant impact. With that data we would require only around 1 000 signatures to mount this attack.

If the b_j are unknown, then, like also done in [10], we only use those equations where $z_{ji} = y_{ji}$ and thus $\langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle = 0$. Then, one can search for the key $\mathbf{s}$ in the kernel of the matrix composed by the $\mathbf{c}_{ji}$. As seen in Figure 4a, $\langle \mathbf{s}_1, \mathbf{c}_{ji} \rangle = 0$ holds for about 15 % of all samples, thus the number of required traces needs to be multiplied by 6.6. Hence, 29 000 and 239 000 signatures are required for A2 and A3, respectively. In the remainder of this paper, we give the signature requirements for both cases of b. We state the requirements with a known b first, followed by the unknown case in parentheses.

5.4 An Attack on Shuffling - Basic Concept

If the elements of $\mathbf{y}_1$ are shuffled after sampling, then the above attack is not directly applicable. To still use it, we first need to do an *un-shuffling*, i.e., we need to re-assign recovered Gaussian samples to their respective index in the signature and thus to the correct $z_i \in \mathbf{z}_1$.

We do that by exploiting the differing (coefficient-wise) distributions of $\mathbf{s}_1\mathbf{c}$ and $\mathbf{y}_1$, they are shown in Figure 4. The distribution of $\mathbf{s}_1\mathbf{c}$, which we denote with $\mathcal{X}_{\mathbf{sc}}$, was estimated using a histogram approach, whereas $\mathbf{y}_1$ follows D_σ^n. Observe that the standard deviation of $\mathbf{y}_1$ is much larger than that of $\mathbf{s}_1\mathbf{c}$. Thus, we can say that $\mathbf{z}_1 \approx \mathbf{y}_1$.

We use this relation as basis of our attack. If we know one particular coefficient y of $\mathbf{y}_1$ but not its position due to shuffling, then we can test all coefficients of the public $\mathbf{z}_1$ for proximity to y. If only a single $z_i \in \mathbf{z}_1$ is "close" to y, then we can assign y to the position of z_i and compute $z_i - y$ to retrieve the value of $(-1)^b \langle \mathbf{s}_1, \mathbf{c}_i \rangle$. As actual metric for closeness, we use $\mathcal{X}_{\mathbf{sc}}(z_i - y)$. Observe that this approach is expected to succeed mostly for large absolute values of y and thus z_i, i.e., in the tail of D_σ. Due to the high dimension $n = 512$, there will be many similar values of y and z_i near the center, thus a unique assignment will not be possible in those cases.

5.5 Attack Details

The previous description of our attack is relatively informal, we now give a more in-depth explanation. For now, consider the case of single-stage shuffling, we adapt the approaches to the two-stage variant later on.

Given two values z_i and y, we define $z_i \sim y$ as the event that z_i and y belong to the same index i in the signature. Without considering knowledge of other processed values, we have a likelihood $\mathrm{P}(z_i \sim y) = \mathcal{X}_{\mathbf{sc}}(z_i - y)$.

When now given the public $\mathbf{z}_1$ and a single sample y of $\mathbf{y}_1$, we can compute, for each $z_i \in \mathbf{z}_1$, $\mathrm{P}(z_i \sim y|\mathbf{z}_1)$. We do that by using Bayes' theorem with uniform prior, i.e., $\mathrm{P}(z_i \sim y|\mathbf{z}_1) = \mathrm{P}(z_i - y) / \sum_{z_j \in \mathbf{z}_1} \mathrm{P}(z_j - y)$. Analogously, for a single z_i and a fully reconstructed but shuffled $\mathbf{y}_1$, we can compute $\mathrm{P}(z_i \sim y_j|\mathbf{y}_1)$.

We perform this analysis on every possible combination of y and z_i. Thus, we compute a likelihood matrix $\mathbf{L} \in (n \times n)$, with $\mathbf{L}_{i,j} = \mathcal{X}_{\mathbf{sc}}(z_i - y_j)$. Afterwards, we apply the Bayesian step to both the columns and the rows of this matrix in order to derive the aforementioned conditional probabilities. Then, we combine both normalized matrices by taking their maximum, i.e., we set $\mathrm{P}(z_i \sim y_j) = \max(\mathrm{P}(z_i \sim y_j|\mathbf{z}_1), \mathrm{P}(z_i \sim y_j|\mathbf{y}_1))$. In other words, we both search for z_i that fit to only one y, and y that fit to only one z_i. Finally, for each $z_i \in \mathbf{z}_1$, we pick the most likely y as $\operatorname{argmax}_{y_j} \mathrm{P}(z_i \sim y_j)$. This shuffling analysis is repeated for each recorded signature.

The previously discussed key-recovery algorithm requires errorless information. Thus, we keep only pairs of (z_i, y) that match with very high probability; we set the threshold to 0.99. Even so, matching errors cannot be entirely excluded. However, a small number of errors can be corrected by gathering slightly more than $n = 512$ equations, and then performing the key-recovery procedure multiple times, each time using a random subset of the collected equations. Alternatively, one could use the lattice-based techniques discussed by Groot Bruinderink et al. [10].

Merging equal *y*. For key-recovery we compute $z_i - y$ for each recovered pair (z_i, y). Here, the actual *index* of y is irrelevant, only the *value* of y needs to be correct. Consequently, if $\mathbf{y}_1$ contains multiple copies of the same value, then they can be treated as a single entity.

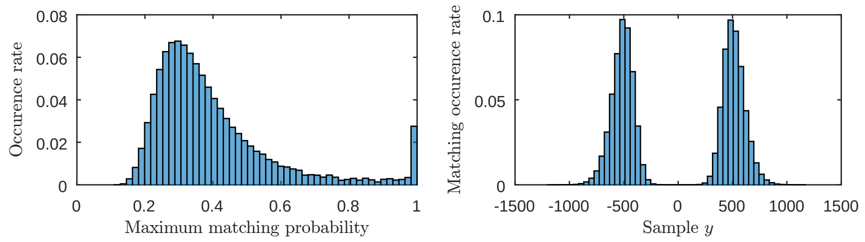
We use this observation as follows. We create a vector $\mathbf{u}$ which contains the unique elements of $\mathbf{y}_1$. We then compute $\mathrm{P}(z_i \sim u_j|\mathbf{u})$. For that, we use the number of times each u_j appears in $\mathbf{u}$ as prior probabilities (instead of the uniform distribution). For $\mathbf{y} \leftarrow D_\sigma$, the average number of unique elements in $\mathbf{y}$ is 377. For $\mathbf{y}' \leftarrow D_{\sigma'}$ only 92 elements are unique on average. Especially in the latter case, the merging of equal y' increases the rate of matches and also decreases the computation time of the subsequent analysis. From now on, we will always use this optimization implicitly.

Note that merging equal values of $\mathbf{z}_1$ is not useful. As already hinted by always using subscripts, each z_i is coupled to one specific $\mathbf{c}_i$, i.e., a negacyclic rotation of the signature part $\mathbf{c}$.

Results on single-stage shuffling. We evaluated our described attack against (single-stage) shuffling by running it with 2^{20} signatures. The results for A1 are shown in Figure 5. 2.5 % of all samples match with a probability of at least 0.99. With this number, only 40 (264) signatures are required to gather $n = 512$ equations. As expected, the successfully matched y lie in the tail of D'_σ (Figure 5b).

For A2 and A3, we do not know the entire $\mathbf{y}_1$ and so did not compute $\mathrm{P}(z_i \sim y_j|\mathbf{y}_1)$. When only using $\mathrm{P}(z_i \sim y|\mathbf{z}_1)$, we can match a proportion of $1.4 \cdot 10^{-4}$ (A2) and $2.2 \cdot 10^{-5}$ (A3) of all samples. This translates to requiring 7 000 (46 000) and 46 000 (301 000) signatures, respectively. The number of expected errors in $n = 512$ equations is well below 1 for all considered adversaries.

When compared to the signature requirements without shuffling, one can observe only a marginal increase. All numbers are well low enough to be practical, thus shuffling once is not an effective countermeasure.



(a) Histogram of maximum matching probabilities.

(b) Histogram of the value of successfully matched y

Fig. 5. Result for the attack on single-stage shuffling, attacker A1

5.6 Adaptation to Two-Stage Shuffling

For two-stage shuffling, the $\mathbf{y}', \mathbf{y}''$ are independently permuted. Thus, we cannot compute any elements of $\mathbf{y}_1$ in straight-forward manner which makes the attack from above not directly applicable. A similar one, however, is still possible; we now state the required modifications. As the sampling (and shuffling) process proceeds in two steps, we also adapt a two-stage approach in the attack.

Assume we are given $\mathbf{z}_1$ and the *shuffled* $\mathbf{y}', \mathbf{y}''$, with $\mathbf{y}_1 = k\mathbf{y}' + \mathbf{y}''$ and hence $\mathbf{z}_1 = k\mathbf{y}' + \mathbf{y}'' + (-1)^b \mathbf{s}_1 \mathbf{c}$. We first aim at finding matching pairs for elements of $\mathbf{z}_1$ and $\mathbf{y}'$. Afterwards, for each pair (z_i, y'), we compute $z_i - ky'$ and then match this difference with the elements of the second vector $\mathbf{y}''$. We now explain the details of this process.

First stage. The first part differs from the previous attack mainly as in we now test the proximity of elements of $\mathbf{z}_1$ to those of $k\mathbf{y}'$. As $\mathbf{z}_1 - k\mathbf{y}' = \mathbf{y}'' + (-1)^b \mathbf{s}_1 \mathbf{c}$, we cannot test proximity with regards to $\mathcal{X}_{\mathbf{sc}}$. Instead, we could use the distribution of an $x = x_1 + x_2$, with $x_1 \leftarrow D_{\sigma'}$ and $x_2 \leftarrow \mathcal{X}_{\mathbf{sc}}$. We denote it as $\mathcal{X}_{\mathbf{sc}+D_{\sigma'}}$. However, as the attacker has (at least partial) knowledge on $\mathbf{y}''$, this would be suboptimal. Hence, we set (with some abuse of notation) $x_2 \leftarrow \mathbf{y}''$, i.e., randomly chosen elements from $\mathbf{y}''$. We call the resulting distribution $\mathcal{X}_{\mathbf{sc}+\mathbf{y}''}$ and use it to fill our likelihood matrix $\mathbf{L}$ with $\mathbf{L}_{i,j} = \mathcal{X}_{\mathbf{sc}+\mathbf{y}''}(z_i - ky'_j)$. The remainder of the analysis, i.e., the Bayesian steps and picking the maximum, are then the same. Finally, all samples matched with probability greater than 0.99 are fed to the second stage of the recovery.

For A2 and A3, we require additional modifications. First, we cannot compute $\mathcal{X}_{\mathbf{sc}+\mathbf{y}''}$, as $\mathbf{y}''$ is only partially known. Instead, we construct a hybrid distribution that merges $D_{\sigma'}$ (for all unknown samples up to the threshold of 47 and 54, respectively) and the known samples of $\mathbf{y}''$. Then, unlike in the single-stage attack, we would also like to compute (or rather estimate) $\mathrm{P}(z_i \sim y'_j | \mathbf{y}')$ despite not having the full $\mathbf{y}'$. We do so by introducing a dummy sample y'_d, which represents all (unknown) samples below the model threshold. Thus, we test if z_i matches with any of the known y'_j or with any element below the threshold. We set the likelihood of y'_d as in (2), the remaining steps are then equivalent to those of A1.

$$\mathrm{P}(z_i \sim y'_d) = \sum_{y=-\mathrm{threshold}}^{\mathrm{threshold}} \mathcal{X}_{\mathbf{sc}+D_{\sigma'}}(z_i - y) \tag{2}$$

Second stage. In the second stage, we test each pair (z_i, y'_i) found in the previous stage with the elements of $\mathbf{y}''$. We do so by computing $z_i - ky'_i$ and then testing for proximity to the elements of $\mathbf{y}''$ with regards to $\mathcal{X}_{\mathbf{sc}}$.

Even for A1, the expected number of matched pairs per signature in the first stage is relatively small. Thus, we cannot compute $\mathrm{P}((z_i - ky'_i) \sim y''_j | (\mathbf{z}_1 - k\mathbf{y}'))$ and are left with $\mathrm{P}((z_i - ky'_i) \sim y''_j | \mathbf{y}'')$. Like in the first stage, A2 and A3 only have partial knowledge of $\mathbf{y}''$. We use the same trick as above and introduce a dummy sample y''_d representing all elements below the modeled threshold of 47 and 55, respectively. Here we use (3) and then again perform the Bayesian step and a filtering of the most probable matches.

$$\mathrm{P}((z_i - ky'_i) \sim y''_d) = \sum_{y=-\mathrm{threshold}}^{\mathrm{threshold}} \mathcal{X}_{\mathbf{sc}}(z_i - ky'_i - y) \tag{3}$$

Results on two-stage shuffling. Like earlier, we evaluated our described attack against two-stage shuffling by running it with 2^{20} signatures. For our ideal adversary A1, we can match 0.26 % of samples in the first stage (with probability greater than 0.99). Out of the found pairs, 0.15 % can also be matched in the second stage. This results in requiring 260 000 (1 550 000) signatures in order to find $n = 512$ equations.

Interestingly, the losses incurred by the restrictions of A2 are relatively small. We can match 0.25 % in the first and 0.15 % of samples in the second stage. With 285 000 (1 880 000), the number of required signatures is virtually identical to the previous case. As to be expected, A3 performs slightly worse. The matching rates decrease to 0.18 % and 0.10 %, respectively. This results in requiring 575 000 (3 800 000) signatures.

Discussion. Apparently, two-stage shuffling can increase the number of required signatures for an attack significantly and thus can be considered an effective countermeasure. For A1, for instance, 260 000 (1 710 000) instead of the previous 40 (264) signatures are required. However, while the given numbers are high, they are still within reach for a dedicated attacker.

This large increase could be explained as follows. For single-stage shuffling, we tested elements from D_σ, with $\sigma \approx 215$, against a distance of $\mathcal{X}_{\mathbf{sc}}$. For the two-stage attack, the ratio of the matched standard deviations is much smaller. For instance, in the second stage we match elements from $D_{\sigma'}$, with $\sigma' \approx 19.5$, against the same $\mathcal{X}_{\mathbf{sc}}$. As a result, the matchable samples are even further out in the tail of $D_{\sigma'}$ and so less frequent than was the case for single-stage shuffling. This also explains the compared to A1 maybe surprisingly small losses of A2 and A3. These adversaries can only recover a small number of samples, but the ones they can find are already in tail $D_{\sigma'}$ and thus more likely to be usable. Obviously, the effect of smaller difference of deviations is amplified by requiring two matching steps. So, we can only rewind shuffling for indizes i where *both* y'_i and y''_i are outliers.

6 Conclusion

Our work shows that shuffling is, at least if done correctly, an effective and cheap countermeasure in the context of lattice-based signatures. However, while it can drastically increase the attack complexity, relying on two-stage shuffling alone might not be enough to protect against attacks on Gaussian samplers. The reported signature requirements for attacks are still practical, at least in the case of a recoverable b. In this regard, recall that we did not use leakage from either multiplication with k and addition of two samples in the Gaussian convolution, the shuffling itself, or from the PRNG. This information can be used to further decrease the number of required signatures. Thus, a mix of countermeasures and reducing the leakage of the sampling algorithm itself is necessary for sufficient protection. Increasing the number of sampling (and shuffling) stages as well as the use of different convolution parameters might also offer better protection. For future work, we plan to further investigate and improve the attack technique. Our goal is to eliminate the impact of an unknown b, i.e., to use the same number of signatures as in the known-b case.

Acknowledgements
This work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 845589 (SCALAS). We would also like to thank Leon Groot Bruinderink for his ideas and valuable input.

References

1. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum Key Exchange - A New Hope. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, 2016.
2. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In *SP 2015*, pages 553–570. IEEE Computer Society, 2015.
3. M. Braithwaite. Experimenting with Post-Quantum Cryptography, July 2016. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.
4. J. A. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden. Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers. In T. Lange, K. E. Lauter, and P. Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 402–417. Springer, 2013.
5. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.
6. L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. NISTIR 8105 DRAFT, Report on Post Quantum Cryptography, February 2016. http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf.
7. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice Signatures and Bimodal Gaussians. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013.
8. N. C. Dwarakanath and S. D. Galbraith. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, 2014.
9. B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. Stochastic Methods. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 15–29. Springer, 2006.
10. L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme. In B. Gierlichs and A. Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, 2016. full version available at http://eprint.iacr.org/2016/300.
11. T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In E. Prouff and P. Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, 2012.
12. J. Jaffe. A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter. In P. Paillier and I. Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 1–13. Springer, 2007.
13. D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*, chapter 3, pages 145–146. Addison-Wesley, 3rd edition, 1998.
14. R. Lindner and C. Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In A. Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011.
15. A. Moradi and G. Hinterwälder. Side-Channel Security Analysis of Ultra-Low-Power FRAM-Based MCUs. In S. Mangard and A. Y. Poschmann, editors, *COSADE 2015*, volume 9064 of *LNCS*, pages 239–254. Springer, 2015.

16. NSA/IAD. CNSA Suite and Quantum Computing FAQ, January 2016. https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm.
17. T. Oder, T. Pöppelmann, and T. Güneysu. Beyond ECDSA and RSA: Lattice-based Digital Signatures on Constrained Devices. In *DAC '14*, pages 110:1–110:6. ACM, 2014.
18. T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced Lattice-Based Signatures on Reconfigurable Hardware. In L. Batina and M. Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 353–370. Springer, 2014. VHDL source code available at http://sha.rub.de/research/projects/lattice.
19. T. Pöppelmann, T. Oder, and T. Güneysu. High-Performance Ideal Lattice-Based Cryptography on 8-Bit ATxmega Microcontrollers. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, 2015.
20. S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and Side Channel Secure Discrete Gaussian Sampling. Cryptology ePrint Archive, Report 2014/591, 2014. http://eprint.iacr.org/2014/591.
21. M.-J. O. Saarinen. Arithmetic Coding and Blinding Countermeasures for Lattice Signatures: Engineering a Side-Channel Resistant Post-Quantum Signature Scheme with Compact Signatures. Cryptology ePrint Archive, Report 2016/276, 2016. http://eprint.iacr.org/2016/276 Note: to appear in Journal of Cryptographic Engineering.
22. B. Schneier. NSA Plans for a Post-Quantum World, August 2015. https://www.schneier.com/blog/archives/2015/08/nsa_plans_for_a.html.
23. strongSwan. strongSwan 5.2.2 Released. https://www.strongswan.org/blog/2015/01/05/strongswan-5.2.2-released.html, 2015.
24. Texas Instruments. MSP432P401R LaunchPad. http://www.ti.com/tool/msp-exp432p401r.