Enhanced Lattice-Based Signatures on Reconfigurable Hardware

Thomas Pöppelmann[1], Léo Ducas[2], and Tim Güneysu[1]

[1] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
[2] University of California, San-Diego

Abstract. The recent BLISS signature scheme showed that lattice-based constructions have evolved to practical alternatives to RSA or ECC. Besides reasonably small signatures with 5600 bits for a 128-bit level of security, BLISS enables extremely fast signing and signature verification in software. However, due to the complex sampling of Gaussian noise with high precision, it is not clear whether this scheme can be mapped efficiently to embedded devices. In particular, the software approach of using large precomputed tables for Gaussian sampling cannot be transferred to constrained computing environments, such as FPGAs with limited memory. In this work we present techniques for an efficient CDT-based Gaussian sampler on reconfigurable hardware involving Peikert's convolution lemma and the Kullback-Leibler divergence. Based on our enhanced sampler design, we provide a first BLISS architecture for Xilinx Spartan-6 FPGAs that integrates fast FFT/NTT-based polynomial multiplication, sparse multiplication, and a Keccak hash function. With on our core a signing operations requires 123 µs on average, using 2,584 slices, 8 BRAMs, and 6 DSPs. Verification takes slightly less with 70 µs.

Keywords: Ideal Lattices, Gaussian Sampling, Digital Signatures, FPGA

1 Introduction and Motivation

Virtually all currently used digital signature schemes rely either on the factoring (RSA) or the discrete logarithm problem (DSA/ECDSA). However, with Shor's algorithm [30] sufficiently large quantum computers can solve these problems in polynomial time what potentially puts billions of devices and users at risk. Although powerful quantum computers will certainly not become available soon, significant resources are definitely spent by various organizations to boost their further development [27]. Also motivated by further advances in classical cryptanalysis (e.g., [4, 5, 18]), it is important to investigate potential alternatives now to have secure constructions and implementations at hand when they are finally needed.

In this work we deal with such a promising alternative, namely the Bimodal Lattice Signature Scheme (BLISS) [9], and specifically address implementation challenges for constrained devices and reconfigurable hardware. First efforts in this direction were made in 2012 by Güneysu et al. [13] (GLP). Their scheme was based on work by Lyubashevsky [21] and tuned for practicability and efficiency in embedded systems. This was achieved by a new signature compression mechanism, a more "aggressive", non-standard hardness assumption, and the decision to use uniform instead of Gaussian noise to hide the secret key contained in each signature via rejection sampling. While GLP allows high performance on low-cost FPGAs [13] and CPUs [14] it later turned out that the scheme is suboptimal in terms of signature size and its claimed security level compared to BLISS. The main reason for this is that Gaussian noise, which is prevalent in almost all lattice-based constructions, allows more efficient, more secure, and also smaller signatures. However, while other techniques relevant for lattice-based cryptography, like fast polynomial arithmetic on ideal lattices received some attention [1, 26, 28], it is currently not clear how efficient Gaussian sampling can be done on reconfigurable and embedded hardware for large standard deviations. Results from

electrical engineering (e.g., [17,33]) are not directly applicable, as they target continuous Gaussians. Applying these algorithms for the discrete case is not trivial (see, e.g., [8] for a discrete version of the Ziggurat algorithm). First progress was recently made by Roy et al. [31] based on work by Galbraith [10] providing results for Gaussian samplers in lattice-based encryption that requires much smaller parameters. For lattice-based digital signature schemes, large tables in straightforward implementations might imply the impression that Gaussian-noise based schemes are a suboptimal choice on constrained embedded systems. A recent example is a microcontroller implementation [7] of BLISS that requires tables for the Gaussian sampler of roughly 40 to 50 KB on an ATxmega64A3. Other lattice-based signatures with explicit reductions to standard lattice problems [11, 20, 23] are also inefficient in terms of practical signature and public key sizes (see [3] for an implementation of [23]). Thus, despite the necessity of improving Gaussian sampling techniques (which is one contribution of this work) BLISS seems to be currently the most promising scheme with a signatures length of 5600 bit length, equally large public keys, and 128-bit of equivalent symmetric security. There surely is some room for theoretical improvement, as suggested by the new compression ideas developed by Bai and Galbraith [2]; one can hope that all those techniques can be combined to further improve lattice based signatures.

Contribution. A first contribution of this work are improved techniques for efficient sampling of Gaussian noise that support parameters required for digital signature schemes such as BLISS and similar constructions. We show how to accelerate the binary search on a cumulative distribution table (CDT) using a shortcut table of intervals and develop an optimal data structure that saves roughly half of the table space by exploiting the properties of the Kullback-Leibler divergence. Furthermore, we apply a convolution lemma [24] for discrete Gaussians that results in even smaller tables of only 2.6 KB for BLISS-I parameters. Based on these techniques we provide an implementation of the BLISS-I parameter set on reconfigurable hardware that is tweaked for performance and offers 128-bit of security[3]. For practical evaluation we compare our improvements for the CDT-based Gaussian sampler to the Bernoulli approach presented in [9]. Our implementation includes an FFT/NTT-based polynomial multiplication (contrary to the schoolbook approach from [13]), more efficient sparse multiplication, and the KECCAK-f[1600] hash function to provide the full picture of the performance that can be achieved by employing latest lattice-based signature schemes on reconfigurable hardware. Our implementation on a Xilinx Spartan-6 FPGA supports up to 8,143 signatures per second using 7,979 LUTs, 7,136 flip-flops and 8 block RAMs and outperforms previous work [13] both in time and area.

2 The Bimodal Lattice Signature Scheme

The most efficient instantiation of the BLISS signature scheme [9] is based on ideal-lattices [22] and operates on polynomials over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. For quick reference, the BLISS key generation, signing as well as verification algorithms are given in Figure 1 and implementation relevant parameters as well as achievable signature and key sizes are listed in Table 1. Note that for the remainder of this work, we will focus solely on BLISS-I. The BLISS key generation basically involves uniform sampling of two small and sparse polynomials $\mathbf{f}, \mathbf{g}$, computation of a certain rejection condition ($N_\kappa(\mathbf{S})$), and computation of an inverse. For signature generation two polynomials $\mathbf{y}_1, \mathbf{y}_2$ of

[3] We will make the VHDL design of our BLISS-I implementation publicly available after conference publication to allow independent verification of our results.

Table 1: Parameters proposals from [9].

Name of the scheme	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**
Security	128 bits	128 bits	160 bits	192 bits
(n, q)	(512,12289)	(512,12289)	(512,12289)	(512,12289)
Secret key densities δ_1, δ_2	0.3 , 0	0.3 , 0	0.42 , 0.03	0.45, 0.06
Gaussian std. dev. σ	215	107	250	271
Weight of the challenge κ	23	23	30	39
Verif. thresholds B_2, B_∞	12872, 2100	11074, 1563	10206,1760	9901, 1613
Repetition rate	1.6	7.4	2.8	5.2
Signature size	5.6kb	5kb	6kb	6.5kb
Secret key size	2kb	2kb	3kb	3kb
Public key size	7kb	7kb	7kb	7kb

length n are sampled from a discrete Gaussian distribution with standard deviation σ. Note that the computation of $\mathbf{a}\mathbf{y}_1$ can still be performed in the FFT-enabled ring $\mathcal{R}_q$ instead of $\mathcal{R}_{2q}$. The result $\mathbf{u}$ is then hashed with the message μ. The output of the hash function is interpreted as sparse polynomial $\mathbf{c}$. The polynomials $\mathbf{y}_{1,2}$ are then used to mask the secret keys $\mathbf{s}_{1,2}$ which are multiplied with the polynomial $\mathbf{c}$ and thus "sign" the hash of the message. In order to prevent any leakage of information on the secret key, rejection sampling is performed and signing might restart. Finally, the signature is compressed and $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ returned. For verification the norms of the signature are first validated, then the input to the hash function is reconstructed and it is checked whether the corresponding hash output matches $\mathbf{c}$ from the signature.

Algorithm KeyGen()

1: Choose $\mathbf{f}, \mathbf{g}$ as uniform polynomials with exactly $d_1 = \lceil \delta_1 n \rceil$ entries in $\{\pm 1\}$ and $d_2 = \lceil \delta_2 n \rceil$ entries in $\{\pm 2\}$
2: $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^t$
3: **if** $N_\kappa(\mathbf{S}) \geq C^2 \cdot 5 \cdot (\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil) \cdot \kappa$ **then restart**
4: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \bmod q$ (**restart** if $\mathbf{f}$ is not invertible)
5: **Return**$(pk = \mathbf{A}, sk = \mathbf{S})$ where $\mathbf{A} = (\mathbf{a}_1 = 2\mathbf{a}_q, q - 2) \bmod 2q$

Alg. Sign$(\mu, pk = \mathbf{A}, sk = \mathbf{S})$

1: $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D_{\mathbb{Z}^n, \sigma}$
2: $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$
3: $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$
4: Choose a random bit b
5: $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$
6: $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$
7: **Continue** with probability $1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2} \right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc} \rangle}{\sigma^2} \right) \right)$ otherwise **restart**
8: $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
9: **Return** $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

Alg. Verify$(\mu, pk = \mathbf{A}, (\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c}))$

1: **if** $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_2 > B_2$ **then** Reject
2: **if** $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_\infty > B_\infty$ **then** Reject
3: **Accept** iff $\mathbf{c} = H\big(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu\big)$

Fig. 1: The Bimodal Lattice Signature Scheme [9].

3 Improving Gaussian Sampling for Lattice-Based Digital Signatures

Target distribution. We recall that the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ is defined by a weight proportional to $\rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$ for all integers x. Our goal is to efficiently sample from that distribution for a constant value $\sigma = 215$ as specified in BLISS-I. This can easily be reduced to sampling from a distribution over $\mathbb{Z}^+$ proportional to $\rho(x)$ for all $x > 0$ and to $\rho(0)/2$ for $x = 0$.

Overview. Gaussian sampling using a large cumulative distribution table (CDT) has been shown to be an efficient strategy for the software implementation of BLISS given in [9]. In this section, we further enhance CDT-based Gaussian sampling for use on constrained devices. For simplicity, we explicitly refer to the parameter set BLISS-I although we remark that our enhancements can be transferred to any other parameter set as well. For performance we first analyze and improve the binary search step to reduce the number of comparisons (cf. Section 3.1). Second, we decrease the size of the precomputed tables. For that we apply a convolution lemma for discrete Gaussians from [24] that enables the use of a sampler with much smaller standard deviation $\sigma' \approx \sigma/8$ what reduces the table size by a factor 8. Moreover, by introducing the statistical notion of the Kullback-Leibler divergence into the CDT approach (cf. Section 3.3), we finally reduce the size of the precomputed table further by roughly a factor of two. Kullback-Leibler is a standard notion in information theory and already played a role in cryptography, mostly in the context of symmetric cryptanalysis [6, 34].

3.1 Binary Search with Shortcut Intervals

The CDT sampling algorithm uses a table $0 = T[0] \leq T[i] \leq \cdots \leq T[S+1] = 1$ to sample from a uniform real $r \in [0, 1)$. The unique result x is obtained from a binary search satisfying that $T[x] \leq r < T[x+1]$ so that each output x has a probability $T[x+1] - T[x]$. For BLISS-I we need a table as large as $S = 2838 \approx 13.2\sigma$ to dismiss only a portion of the tail less than 2^{-128}. As a result, the naive binary search would require $\log_2 S = 11.6$ comparisons on average.

As an improvement we propose to combine the binary search with a hash map based on the first bits of r to narrow down the search interval in a first step. For the given parameters and memory alignment reasons, we choose the first byte of r for this hash map: the unique $v \in \{0 \dots 255\}$ such that $v/256 \leq r < (v+1)/256$. This table of intervals has length 256 and each entry $T[v]$ encodes the smallest interval (a, b) such that $T[a] \geq v/256$ and $T[b] \geq (v+1)/256$. With this approach, the search can be directly reduced to the interval (a, b) resulting in 1.6 comparisons on average.

3.2 Reducing Precomputed Data by Gaussian Convolution

Given that x_1, x_2 are variables from continuous Gaussian distributions with variances σ_1^2, σ_2^2, then their combination $x_1 + cx_2$ is Gaussian with variance $\sigma_1^2 + c^2\sigma_2^2$ for any c. While this is not generally the case for discrete Gaussians, there exists similar convolution properties under some smoothing condition as obtained from [24, 25]:

Corollary 1 (of Thm. 3.3 from [24]). *For any nonzero integer vector* $\mathbf{z} \in \mathbb{Z}^m$ *such that* $\gcd(\mathbf{z}) = 1$*, if for all* $i \leq n$ $\sigma_i \geq \sqrt{1/\pi} \cdot \|\mathbf{z}\|_\infty \cdot \eta_\epsilon(\mathbb{Z})$*; then the distribution of* $x = \sum_i z_i x_i$ *where* $x_i \leftarrow D_{\mathbb{Z},\sigma_i}$ *is at statistical distance less than* $O(n\epsilon)$ *from* $D_{\mathbb{Z},\sigma}$ *where* $\sigma^2 = \sum_i z_i^2 \sigma_i^2$*.*

For our parameters, we chose $\mathbf{z} = (1, 8)$, and $\sigma' = \sigma/\sqrt{1+8^2} \approx 26.66$ so that the smoothness condition is given for $\epsilon = 2^{-128}$ and $\eta_\epsilon(\mathbb{Z}) \approx 5.35$ (see [24] for the definition of the smoothing parameter η). Due to σ' the size of the precomputation table reduces by a factor of 8 at the price of sampling twice. However, the running time does not double in practice since the enhancement based on the shortcut intervals reduces the number of necessary comparisons to 0.3 on average: for a majority of first byte v; the interval length $b - a$ is reduced to 1; and x is determined without any comparison.

3.3 Applications of the Kullback-Leibler Divergence

We now present the notion of Kullback-Leibler (KL) divergence that is later used to further reduce the table size. Detailed proofs of following lemmata are given in the Appendix A.2.

Definition 1 (Kullback-Leibler Divergence). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions over a common countable set Ω, and let $S \subset \Omega$ be the support of $\mathcal{P}$. The Kullback-Leibler divergence, noted D_{KL} of $\mathcal{Q}$ from $\mathcal{P}$ is defined as:*

$$D_{KL}(\mathcal{P}\|\mathcal{Q}) = \sum_{i \in S} \ln\left(\frac{\mathcal{P}(i)}{\mathcal{Q}(i)}\right)\mathcal{P}(i)$$

with the convention that $\ln(x/0) = +\infty$ for any $x > 0$.

The Kullback-Leibler divergence shares many useful properties with the more usual notion of statistical distance. First, it is additive so that $D_{\mathrm{KL}}(\mathcal{P}_0 \times \mathcal{P}_1 \| \mathcal{Q}_0 \times \mathcal{Q}_1) = D_{\mathrm{KL}}(\mathcal{P}_0\|\mathcal{Q}_0) + D_{\mathrm{KL}}(\mathcal{P}_1\|\mathcal{Q}_1)$ and, second, non-increasing under any function $D_{\mathrm{KL}}(f(\mathcal{P})\|f(\mathcal{Q})) \leq D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q})$ (see Lemmata 3 and 4 of Appendix A.2). An important difference though is that it is not symmetric.

Choosing parameters so the theoretical distribution $\mathcal{Q}$ is at KL-divergence about 2^{-128} from the actually sampled distribution $\mathcal{P}$, the next lemma will let us conclude[4] that if the ideal scheme $\mathcal{S}^{\mathcal{Q}}$ has about 128 bits of security, so has the implemented scheme $\mathcal{S}^{\mathcal{P}}$.

Lemma 1 (Bounding Success Probability Variations). *Let $\mathcal{E}^{\mathcal{P}}$ being an algorithm making at most q queries an oracle sampling from a distribution $\mathcal{P}$ and outputting a bit. Let $\epsilon \geq 0$, and $\mathcal{Q}$ be a distribution $D_{KL}(\mathcal{P}\|\mathcal{Q}) \leq \epsilon$. Let x (resp. y) denote the probability that $\mathcal{E}^{\mathcal{P}}$ (resp. $\mathcal{E}^{\mathcal{Q}}$) outputs 1. Then, $|x - y| \leq 1/2\sqrt{q\epsilon}$.*

In certain cases, the KL-divergence can be as small as the square of the statistical distance. For example, noting $\mathcal{B}_c$ the Bernoulli variable that returns 1 with probability c, we have $D_{\mathrm{KL}}(\mathcal{B}_{\frac{1-\epsilon}{2}}\|\mathcal{B}_{\frac{1}{2}}) \approx \epsilon^2/2$. In such a case, one requires $q = O(1/\epsilon^2)$ samples to distinguish those two distribution with constant advantage; that is we can prove a much better security using KL-divergence than statistical distance for which the typical argument would only prove security up to $q = O(1/\epsilon)$ queries.

Intuitively, statistical distance is the sum of absolute errors, while KL-divergence is about the sum of squared relative errors.

Lemma 2 (Kullback-Leibler divergence for bounded relative error). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions of same support S. Assume that for some $\delta(i) \in (0, 1/4)$ we have a relative error bound: $\forall i \in S, |\mathcal{P}(i) - \mathcal{Q}(i)| \leq \delta(i)\mathcal{P}(i)$. Then $D_{KL}(\mathcal{P}\|\mathcal{Q}) \leq 2\sum_{i \in S} \delta(i)^2\mathcal{P}(i)$.*

[4] Apply the lemma to an attacker with success probability 1/2 against $\mathcal{S}^{\mathcal{P}}$ and number of queries $< 2^{126}$ (amplifying success probability by repeating the attack if necessary), and deduce that it also succeeds against $\mathcal{S}^{\mathcal{Q}}$ with probability 1/4.

Using floating-point representation, it seems now possible to halve the storage ensuring a relative precision of 64 bits instead of an absolute precision of 128 bits. We exploit this idea for hardware as follows.

3.4 CDT Sampling with Reduced Table Size

Using the table $0 = T[0] \leq T[i] \leq \cdots \leq T[S+1] = 1$, the output of a CDT sampler follows the distribution $\mathcal{P}$ with $\mathcal{P}(i) = T[i+1] - T[i]$. When applying the results from KL-divergence obtained above, the relative error of $T[i+1] - T[i]$ might be significantly larger than the one of $T[i]$. This is particularly true for the tail, where $T[i] \approx 1$ but $\mathcal{P}(i)$ is very small. A simple workaround is to reverse the order of the table so that $1 = T[0] \geq T[i] \geq \cdots \geq T[S+1] = 0$ with a slight modification of the algorithm so that $\mathcal{P}(i) = T[i] - T[i+1]$. With this trick, the subtraction only increase the relative error of roughly a factor of σ.

We aim to have a variable precision in the table $T[i]$ so that $\delta(i)^2\mathcal{P}(i)$ is about constant around $2^{-128}/|S|$ as suggested by Lemma 2 while $\delta(i)$ denotes the relative error $\delta(i) = |\mathcal{P}(i) - \mathcal{Q}(i)|/\mathcal{P}(i)$. As a trade-off between optimal variable precision and hardware efficiency, we propose the following data-structure. We define 9 tables $M_0 \dots M_8$ of bytes for the mantissa with respective lengths $\ell_0 \geq \ell_1 \geq \cdots \geq \ell_8$ and another byte table E for exponents, of length ℓ_0. The value $T[i]$ is defined as

$$T[i] = 256^{-E[i]} \cdot \sum_{k=0}^{8} 256^{-(k+1)} \cdot M_k[i]$$

where $M_k[i]$ is defined as 0 when the index is out of bound $i \geq \ell_k$. In other term, the value of $T[i]$ is stored with $p(i) = 9 - \min\{k | \ell_k > i\}$ bytes of precisions. More precisely, lengths are defined as $[\ell_0, \dots, \ell_8] = [352, 352, 317, 293, 269, 240, 197, 173, 112]$ so that we store at least two bytes for each entry up to $i < 352$, three bytes up to $i < 317$ and so forth. Note that no actual computation is involved in constructing $T[i]$ following the plain CDT algorithm.

For evaluation, we used the closed formula for KL-divergence and measured $D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q}) \approx 2^{-126}$. The storage requirements of this table is computed by $2\ell_0 + \ell_1 + \cdots + \ell_8 \approx 2.6$ KB. The straightforward CDF approach requires each entry up to $i < 352$ to be stored with $128 + \log_2 \sigma$ bits of precisions and thus requires a total of 5.5 KB. The storage requirement are graphically depicted by the area under the curves in the top-right quadrant of Figure 2.

4 Implementation on Reconfigurable Hardware

In this section we provide details on our implementation of the BLISS-I signature scheme on a Xilinx Spartan-6 FPGA. We include the enhancements from the previous section to achieve a design that is tweaked for high-performance at moderate resource costs.

4.1 Gaussian Sampling

To evaluate and validate our results in practice, we compare our enhanced CDT sampler with the efficiency of a Bernoulli sampler proposed in previous work [9].

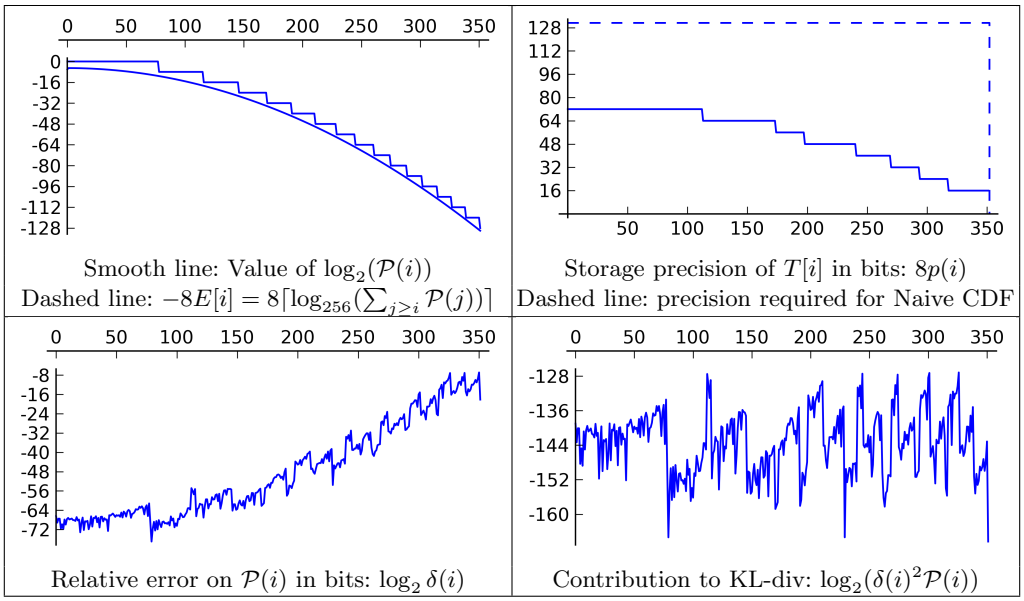

Fig. 2: Data of our optimized CDT sampler for discrete Gaussian of parameter $\sigma' \approx 26.66$

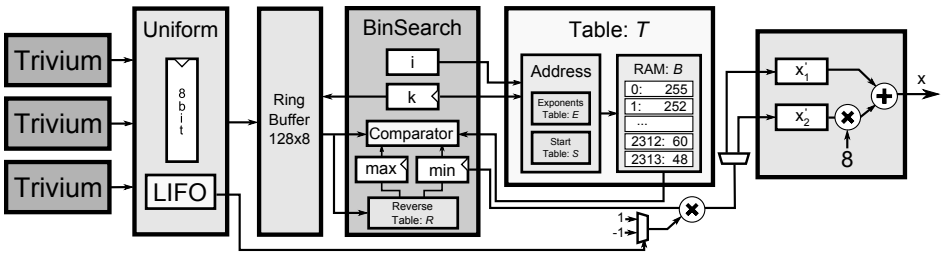

Fig. 3: Block diagram of the CDT sampler which generates two samples x'_1, x'_2 of standard deviation $\sigma' \approx 26.66$ which are combined to a sample $x = x'_1 + 8x'_2$ with standard deviation $\sigma = 215$. The sampling is performed using binary search on the size optimized Table T.

Enhanced CDT Sampling. Along the lines of the previous section our hardware implementation operates on bytes in order to use the 1000x8-bit mode of operation of the Spartan-6 block RAMs. The design of our CDT sampler is depicted in Figure 3. The `BinSearch` component performs a binary search on the table T as described in Section 3.4 for a random byte vector r to find a c such that $T[c] \geq r > T[c+1]$. It accesses T byte-wise and thus $T_j[i] = M_{j-E[i]}[i]$ denotes the entry at index $i \in (0, 352)$ and byte $j \in (0, 24)$. When starting a sampling operation the pointer registers `min` and `max` are initialized with the values stored at position $r[0]$ in the reverse interval table realized as 256x16-bit single port distributed ROM (7 bits for the minimum and 9 bits for the maximum). The index of the middle element of the search radius is $i =$ (`min`+`max`)/2. In case $T_j[i] > r_j$ we set (`min`$= i, i =$ (i+`max`)/2,`max`=`max`). Otherwise, for $T_j[i] < r_j$ we set ($i =$ (`min`+i)/2,`min`=`min`,`max`$= i$) until `max`-`min`< 2. In case of $T_j[i] = r_j[i]$ we increase $j = j + 1$ and thus compare the next byte.

While the binary search is a standard algorithm, the interesting implementation problem is to realize the size reductions and table accesses when proving the interface to $T_j[i]$ in the `Table`

component. The exponent table E is used to determine how many bytes the mantissa needs to be shifted to the right. For a request $T_j[i]$ zero is returned in case $j - E[i] \notin (0, 8)$ or $i > \ell_{j-E[i]}$ and the actual memory address a in the block memory B holding $M_0 \dots M_8$ consecutively is computed as $a = s_{j-E[i]} + i$ where we store the start addresses of each byte group in an additional table S where $[s_0, \dots, s_8] = [0, 353, 706, 1024, 1318, 1588, 1829, 2027, 2201, 2314]$. For random byte generation we used three instantiations of the Trivium stream cipher (each `Trivium` instantiation outputs one bit per clock cycle) to generate a uniformly random byte every third clock cycle and store spare bits in a `LIFO` for later use as sign bits. The random values r_j are stored in a 128x8 bit ring buffer realized as simple dual-port distributed RAM. The idea is that the sampler may request a large number of random bytes in the worst-case but usually finishes after one or two comparisons due to the lazy search. As the `BinSearch` component keeps track of the maximum number of accessed random bytes, it allows the `Uniform` sampler to refresh only the used $\max(j)+1$ bytes in the buffer. In case the buffer is empty, we stop the Gaussian sampler until a sufficient amount of randomness becomes available. In order to compute the final sample x we determine sign bits of two samples x'_1, x'_2 and finally output $x = x'_1 + 8x'_2$.

From an implementation point of view, the sampler should meet the timing requirements of the remaining components ($100 - 140$ MHz). To achieve this frequency range, a comparison in the binary search step could not be performed in one cycle due to the excessive number of tables and range checks involved. We therefore allowed two cycles per search step which are carefully balanced. For example, we precomputed the indices $i' = (\texttt{min}+i)/2$ and $i'' = (i+\texttt{max})/2$ in the cycle prior to a comparison to relax the critical paths. We further merged the block memory B (port A) and the exponent table E (port B) into three 9k block memories and optimized the memory alignment accordingly (B at index 0 to 2313 and E is placed at 2560 to 2912). Note also that we are still accessing the table only every two clock cycles which would enable another sampler to operated on the same table using time-multiplexing.

Bernoulli Approach. In [9] Ducas et al. proposed an efficient Gaussian sampling algorithm which can be used to lower the size of precomputed tables to $\lambda \log_2(2.4\tau\sigma^2)$ bits without the need for long-integer arithmetic and with low entropy consumption ($\approx 6 + 3\log_2 \sigma$). A detailed description and additional background on the sampler is contained in Appendix A.1. The general advantage of this sampler is a new technique to reduce the probability of rejections by first sampling from an (easy to sample) intermediate distribution and then from the target distribution.

The block diagram of the the implemented sampler is given in Figure 4. In the $\texttt{D}^+_{\sigma_2}$ component a $x \in D^+_{\sigma_2}$ is sampled according to Algorithm 2. However, on-the-fly construction of the binary distribution of $\rho_{\sigma_2}(\{0, \dots, j\}) = 1.1001000010000001...$ (see Alg. 2 of Appendix A.1) is not necessary as we use two 64-bit shift registers (LUTM) to store the expansion precomputed up to a precision of 128 bits. Uniformly random values $y \in \{0, \dots, k-1\}$ are sampled in the `Uniform` component using rejection sampling (with $\frac{3}{255}$ the probability of a rejection is low). The pipelined `BerInput` component takes a (y, x) tuple as input and computes $t = kx$ and outputs $z = t + y$ as well as $j = y(y + 2t)$. While z is retained in a register, the $\texttt{B}_{\texttt{exp(-x/f)}}$ module evaluates the Bernoulli distribution of $b \leftarrow B_{\exp(-j/2\sigma^2)}$. Only if $b = 1$ the value z is passed to the output and discarded otherwise. The evaluation of $B_{\exp(-x/f)}$ requires independent evaluations of Bernoulli variables. Sampling from $\mathcal{B}_c$ is easy and can be done by just evaluating $s < c$ for a uniformly random $s \in [0, 1)$ and a precomputed c. The precomputed tables $c_i = \exp(-2^i/f)$ for $0 \le i \le l, f = 2\sigma^2$ where l is $\lceil \log_2(\max(j)) \rceil$ are stored in a distributed RAM. The $\texttt{B}_{\texttt{exp(-x/f)}}$ module (Algorithm 1)

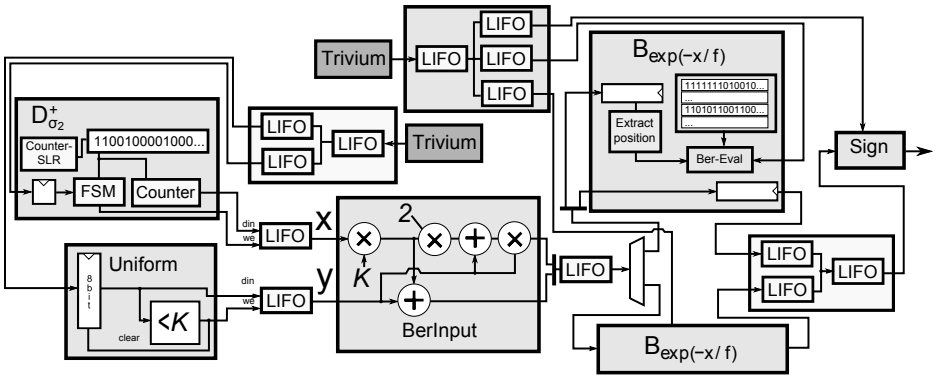

Fig. 4: Block diagram of the Bernoulli sampler using two instantiations of Trivium as PRNG and two `B_exp(-x/f)` components (only one is shown in more detail).

then searches for one-bit positions u in j and evaluates the Bernoulli variable B_{c_u}. This is done in a lazy manner so that the evaluation aborts when the first bit has been found that differs between a random s and c. This techniques saves randomness and also runtime. As the chance of rejection is larger for the most significant bits we scan them first in order to abort as quickly as possible. As the last step the `Sign` component samples a sign bit and performs a rejection in case $z = 0$.

The Bernoulli sampler is suitable for hardware implementation as most operations work on single bits (mostly comparisons) only. However, due to the non-constant time behavior of rejection sampling we had to introduce buffers between each element (see Figure 4) to allow parallel execution and maximum utilization of every component. This includes the distribution and buffering of random bits. In order to reduce the impact of buffering on resource consumption we included Last-In-First-Out (LIFO) buffers that solely require a single port RAM and a counter as the ordering of independent random elements does not need to be preserved by the buffer (what would be the case with a FIFO). For maximum utilization we have evaluated optimal combinations of sub-modules and finally implemented two `B_exp(-x/f)` modules fed by two instantiations of the Trivium stream cipher to generate pseudo random bits. A detailed analysis is given in Section 5.

4.2 Signing and Verification Architecture

The architecture of our implementation of a high-speed BLISS signing engine is given in Figure 5. Similar to the GLP design [13] we implemented a two stage pipeline where the polynomial multiplication $\mathbf{a}\mathbf{y}_1$ runs in parallel to the hashing $H(\mathbf{a}\mathbf{y}_1 + \mathbf{y}2, \mu)$ and sparse multiplication $\mathbf{z}_{1,2} = \mathbf{s}_{1,2}\mathbf{c} + \mathbf{y}_{1,2}$[5]. For polynomial multiplication [1,26,28] of $\mathbf{a}\mathbf{z}_1$ we rely on a publicly available FFT/NTT-based polynomial multiplier [16] (`PolyMul`). The public key $\mathbf{a}$ is stored already in NTT format so that only one forward and one backward transform is required. The multiplier also instantiates either the Bernoulli or the CDT Gaussian sampler (configurable by a VHDL `generic`) and an intermediate FIFO for buffering. When a new triple $(\mathbf{a}\mathbf{y}_1, \mathbf{y}_1, \mathbf{y}_2)$ is available the data is transferred into the block memories `BRAM-U`, `BRAM-Y1` and `BRAM-Y2` and the small polynomial $\mathbf{u} = \zeta\mathbf{a}_1\mathbf{y}_1 + \mathbf{y}_2$ is computed

[5] Another option would be a three stage pipeline with an additional buffer between the hashing and sparse multiplication. As a tradeoff this would allows to use a slower and thus more area efficient hash function but also imply a longer delay and require pipeline flushes in case of an accepted signature.

on-the-fly and stored in `BRAM-U` for later use. The lower order bits $\lfloor \mathbf{u} \rceil_d \bmod p$ of $\mathbf{u}$ are saved in the `RAM-U`. As random oracle we have chosen the KECCAK-$f[1600]$ hash function for its security and speed in hardware [19, 29]. A configurable hardware implementation[6] is provided by the KECCAK project and the `mid-range` core is parametrized so that the KECCAK state it split into 16 pieces ($Nb = 16$). To simplify control logic and padding we just hash multiples of 1024 bit blocks and rehash in case of a rejection. Storage of the state of the hash function after hashing the message (and before hashing $\lfloor \mathbf{u} \rceil_d \bmod p$) would be possible but costly due to the state size of KECCAK. After hashing the `ExtractPos` component extracts the κ positions of $\mathbf{c}$ which are one from the binary hash output and stores them in the 23x9-bit memory `RAM-Pos`.

For the computation of $\mathbf{z}_1' = \mathbf{s}_1\mathbf{c}$ and $\mathbf{z}_2' = \mathbf{s}_2\mathbf{c}$ we then exploited that $\mathbf{c}$ only has $\kappa = 23$ one coefficients. Moreover, only $d_1 = \lceil \delta_1 n \rceil = 154$ coefficients in $\mathbf{s}_1$ are ± 1 and $\mathbf{s}_2$ has d_1 entries in ± 2 where the first coefficient is from $\{-1, 0, 3\}$. The simplest and also best suited algorithm for sparse polynomial multiplication is the row- or column-wise schoolbook algorithm. While row-wise multiplication would benefit from the sparsity of $\mathbf{s}_{1,2}$ *and* $\mathbf{c}$, more memory accesses are necessary to add and store inner products. Since memory that has more than two ports is extremely expensive, this also prevents efficient and configurable parallelization. As a consequence, our implementation consists of a configurable number of cores (C) which perform column-wise multiplication to compute $\mathbf{z}_1$ and $\mathbf{z}_2$, respectively. Each core stores the secret key (either $\mathbf{s}_1$ or $\mathbf{s}_2$) efficiently in a distributed RAM and accumulates inner products in a small multiply-accumulate unit (`MAC`). Positions of $\mathbf{c}$ are fed simultaneously into the cores. Another advantage of our approach is that we can compute the norms and scalar products for rejection sampling parallel to the sparse multiplication. In Figure 5 a configuration with $C = 2$ is shown for simplicity but our experiments show that $C = 8$ leads to an optimal trade-off between speed and resource consumption. Our verification engine uses only the

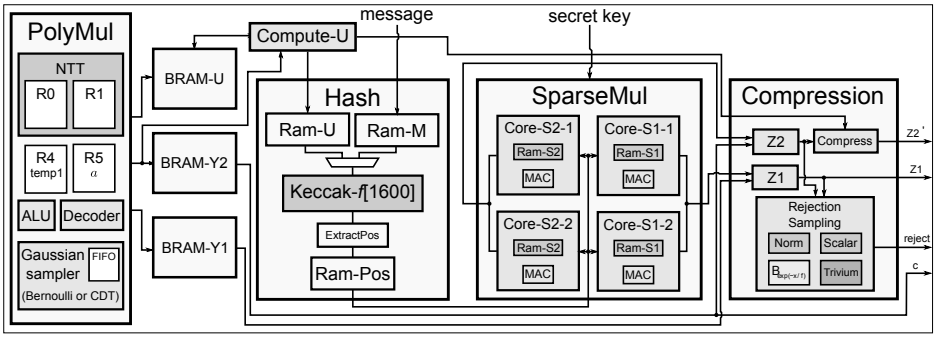

Fig. 5: Block diagram of the implemented BLISS-I signing engine.

`PolyMul` (without a Gaussian sampler) and the `Hash` component and is thus much more lightweight compared to signing. The polynomial $\mathbf{c}$ stored as (unordered) positions is expanded into a 512x1-bit distributed RAM and the input to the hash function is computed in a pipelined manner when `PolyMul` outputs $\mathbf{az}_1$.

[6] See `http://keccak.noekeon.org/mid_range_hw.html` for more information on the core.

5 Results and Comparison

Our results below were obtained post place-and-route (PAR) on a Spartan-6 LX25 (speed grade -3) with Xilinx ISE 14.6.

Gaussian Sampling. Detailed results on area consumption and timing of our two Gaussian sampler designs are given in Table 2. The results show that the enhanced CDT sampler consumes less logic resources than the Bernoulli sampler at the cost of three 9K block memories to store the tables E and B. This is a significant improvement in terms of storage size compared to a naive implementation without the application of the Kullback-Leibler divergence and Gaussian convolution. A standard CDT implementation would require at least $\sigma\tau\lambda = 363264$ bits (equivalent to ≈ 45 9K block Rams) for the defined parameters matching a standard deviation $\sigma = 215$, tailcut $\tau = 13.2$ and precision $\lambda = 128$. The Bernoulli sampler does not require any block RAMs but a large part of the logic consumption can be attributed to the additional buffers to compensate for possible rejections. As illustrated in Figure 4, we feed the Bernoulli sampler with the pseudo-random output of two Trivium instances. For one sample, 33 random bits are required where 12% of the random bits are consumed by the D_{σ_2} module, 42% by the uniform sampler, 43% by both $\mathtt{B_{exp(-x/f)}}$ units and 2.8% for the final sign determination and zero rejection.

With respect to the averaged performance, 7.8 and 18.5 cycles are required by the CDT and the Bernoulli sampler to provide one sample, respectively. We remark that we therefore successfully achieved the goal to reduce the table size of the CDT sampler without sacrificing performance. Another implementation of Gaussian sampling is due to Roy et al. [31]. However, they target a small standard deviation of $\sigma = 3.3$ which required for public key encryption using the Knuth-Yao algorithm (see [10] for more details). Their sampler is area-efficient (47 slices on a Virtex-5) but requires 17 clock cycles for one sample. The discrete Ziggurat [8] performs well in software but does not seem to be a good target for a hardware implementation due to its infrequent rejection sampling operations and its costly requirement on high precision floating point arithmetic.

BLISS Operations. Results for the BLISS signing and verification engine can be found in Table 2 including averaged cycle counts for successfully producing a signature. Please note that 1.6 trials are necessary on average using the BLISS-I parameter set to generate a signature with each trial taking roughly 10k cycles. To evaluate the impact of the sampler used in the design, we instantiated two signing engines of which one employs a CDT sampler and the other one two Bernoulli samplers to match the speed of the multiplier. For a similar performance of roughly 8,000 signing operations per second, the signing instance based on the Bernoulli sampler has a significantly higher resource consumption (about 400 extra slices). Due to the two pipeline stages involved, the runtime of both instances is determined by max(Cycles(`PolyMul`, Cycles(`Hash`) + Cycles(`SparseMul`)) where where the rejection sampling in `Compression` is performed in parallel. Further design space exploration (e.g., evaluating the impact of a different number of parallel sparse multiplication operations or a faster configuration of KECCAK) always identified the `PolyMul` component as performance bottleneck or did not provide significant savings in resources for reduced versions. The verification runtime is determined by Cycles(`PolyMul`+Cycles(`Hash`)) as no pipelining is used and `PolyMul` is slightly faster than for signing as no Gaussian sampling is needed.

Comparison In comparison with the GLP implementation from [13], the design of this work achieves higher throughput with a lower number of block RAMs and DSPs. The structural ad-

Configuration and Operation	Slices	LUT /FF	BRAM18/ DSP	MHz	Cycles	Operations per second (output)
BLISS-I Signing (CDT, $C = 8$)	2,584	7,979/7,136	8/6	132	≈16,210	≈8,143 (signature)
BLISS-I Signing (Dual-Bernoulli, $C = 8$)	2,960	9,029/8,562	6.5/8	131	≈16,210	≈8,081 (signature)
BLISS-I Verification	1,727	5,275/4,488	4.5/3	142	9,835	14,438 (valid/invalid)
Standalone CDT sampler	274	922/1,094	1.5/0	139	≈7.8	≈17,820,512 (sample)
Standalone Bernoulli sampler	416	1,178/1,183	0/1	138	≈18.5	≈7,459,459 (sample)
`PolyMul` (CDT)	1,133	3,321/3,336	7.5/1	129	9,429	13,681 ($\mathbf{a} \cdot \mathbf{y}_1$)
`Hash` ($Nb = 16$)	752	2,461/2,134	0/0	149	1,931	77,162 ($\mathbf{c}$)
`SparseMul` ($C = 1$)	64	162/125	0/0	274	15,876	17,258 ($\mathbf{c} \cdot \mathbf{s}_{1,2}$)
`SparseMul` ($C = 8$)	308	918/459	0/0	267	2,436	109,605 ($\mathbf{c} \cdot \mathbf{s}_{1,2}$)
`SparseMul` ($C = 16$)	628	1847/810	0/0	254	1,476	172,086 ($\mathbf{c} \cdot \mathbf{s}_{1,2}$)
`Compression`	1,230	3,851/3,049	3/0	151	-	parallel to `SparseMul`

Performance and resource consumption of the full BLISS-I signing engine using the CDT sampler or two parallel Bernoulli samplers (Dual-Bernoulli) on the Spartan-6 LX25 for a small 1024 bit message. Additionally we provide area and performance data for the samplers and some sub modules. Note that the final slice, LUT, and FF counts of the signing engine cannot directly be computed as the sum of the sub modules due to cross module optimizations, timing optimization, and additional control logic between modules. Signing and sampling is non-deterministic so we give average cycle counts (≈) based on our simulations.

Table 2: Performance and resource consumption of the full BLISS-I

vantage of BLISS is a smaller polynomial modulus (GLP: $q = 8383489$/BLISS-I: $q = 12289$), less iterations necessary for a valid signature (GLP: 7/BLISS-I: 1.6), and a higher security level (GLP: 80 bit/BLISS-I: 128 bit). Furthermore and contrary to [13], we remark that our implementation takes the area costs and timings of a hash function (KECCAK) into account. In summary, our implementation of BLISS is superior to [13] in almost all aspects. In addition to that Glas et al. [12]

Table 3: Signing or verification speed of comparable signature scheme implementations.

Operation	Security	Algorithm	Device	Resources	Ops/s
GLP [sign] [13]	80	GLP($q = 8383489, n = 512$)	XC6SLX16	7465 LUT/ 8993 FF/ 28 DSP/ 29.5 BRAM18 @270/162 MHz	931
GLP [ver] [13]	80	GLP($q = 8383489, n = 512$)	XC6SLX16	6225 LUT/ 6663 FF/ 8 DSP/ 15 BRAM18 @272/158 MHz	998
RSA [sign] [32]	103	RSA-2048; private key	XC5VLX30T-1	3237 LS/ 17 DSPs	89
ECDSA [sign] [12]	128	Full ECDSA; secp256r1	XC5VLX110T	32299 LUT/FF pairs	139
ECDSA [ver] [12]	128	Full ECDSA; secp256r1	XC5VLX110T	32299 LUT/FF pairs	110

report a vehicle-to-X communication accelerator based on an ECDSA signature over 256-bit prime fields. With respect to this, our BLISS implementation shows again higher performance at less resource cost. Other ECC implementations over 256-bit prime fields (e.g., such as [15] on Xilinx Virtex-4) only implement the point multiplication operation and not the full ECDSA protocol; hence we did not take them into account for a fair comparison. Finally, a fast RSA-2048 core was presented for Virtex-5 devices in [32] which requires more logic/DSPs and provides significantly lower performance (11.2 ms per operation) than our lattice-based signature instance.

References

1. A. Aysu, C. Patterson, and P. Schaumont. Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In *HOST*, pages 81–86. IEEE, 2013.
2. S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. *IACR Cryptology ePrint Archive*, 2013:838, 2013.
3. R. E. Bansarkhani and J. Buchmann. Improvement and efficient implementation of a lattice-based signature scheme. *Selected Areas in Cryptography, SAC 2013, Burnaby, British Columbia, Canada, August 14-16, to appear*, 2013. `http://eprint.iacr.org/2013/297.pdf`.
4. R. Barbulescu. Selecting polynomials for the function field sieve. Cryptology ePrint Archive, Report 2013/200, 2013. `http://eprint.iacr.org/2013/200`.
5. R. Barbulescu, P. Gaudry, A. Joux, and E. Thom. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *CoRR*, abs/1306.4244, 2013.
6. C. Blondeau and B. Gérard. On the data complexity of statistical attacks against block ciphers (full version). Cryptology ePrint Archive, Report 2009/064, 2009. `http://eprint.iacr.org/2009/064`.
7. A. Boorghany and R. Jalili. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. *IACR Cryptology ePrint Archive*, 2014:78, 2014.
8. J. 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. *Selected Areas in Cryptography, SAC 2013, Burnaby, British Columbia, Canada, August 14-16, to appear*, 2013. `http://eprint.iacr.org/2013/510.pdf`.
9. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal gaussians. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Berlin, Germany.
10. S. Galbraith and N. Dwarakanath. Efficient sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *preprint*, 2013. `http://www.math.auckland.ac.nz/~sgal018/gen-gaussians.pdf`.
11. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.
12. B. Glas, O. Sander, V. Stuckert, K. D. Müller-Glaser, and J. Becker. Prime field ECDSA signature processing for reconfigurable embedded systems. *Int. J. Reconfig. Comp.*, 2011, 2011.
13. 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, Leuven, Belgium, Sept. 9–12, 2012. Springer, Berlin, Germany.
14. T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe. Software speed records for lattice-based signatures. In P. Gaborit, editor, *PQCrypto*, volume 7932 of *LNCS*, pages 67–82. Springer, 2013.
15. T. Güneysu and C. Paar. Ultra high performance ECC over NIST primes on commercial FPGAs. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 62–78, Washington, D.C., USA, Aug. 10–13, 2008. Springer, Berlin, Germany.
16. T. Güneysu and T. Pöppelmann. Towards practical lattice-based public-key encryption on reconfigurable hardware. *Selected Areas in Cryptography, SAC 2013, Burnaby, British Columbia, Canada, August 14-16, to appear*, 2013. `http://www.sha.rub.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf`.
17. R. Gutierrez, V. Torres-Carot, and J. Valls. Hardware architecture of a Gaussian noise generator based on the inversion method. *IEEE Trans. on Circuits and Systems*, 59-II(8):501–505, 2012.
18. A. Joux. A new index calculus algorithm with complexity $l(1/4 + o(1))$ in very small characteristic. Cryptology ePrint Archive, Report 2013/095, 2013. `http://eprint.iacr.org/2013/095`.
19. B. Jungk and J. Apfelbeck. Area-efficient FPGA implementations of the SHA-3 finalists. In P. M. Athanas, J. Becker, and R. Cumplido, editors, *ReConFig*, pages 235–241. IEEE Computer Society, 2011.
20. V. Lyubashevsky. Lattice-based identification schemes secure under active attacks. In R. Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 162–179, Barcelona, Spain, Mar. 9–12, 2008. Springer, Berlin, Germany.
21. V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755, Cambridge, UK, Apr. 15–19, 2012. Springer, Berlin, Germany.
22. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

23. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718, Cambridge, UK, Apr. 15–19, 2012. Springer, Berlin, Germany.
24. D. Micciancio and C. Peikert. Hardness of SIS and LWE with small parameters. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Berlin, Germany.
25. C. Peikert. An efficient and parallel Gaussian sampler for lattices. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 80–97, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
26. T. Pöppelmann and T. Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In A. Hevia and G. Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 139–158, Santiago, Chile, Oct. 7–10, 2012. Springer, Berlin, Germany.
27. S. Rich and B. Gellman. NSA seeks quantum computer that could crack most codes. *The Washington Post*, 2013. `http://wapo.st/19DycJT`.
28. S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact hardware implementation of Ring-LWE cryptosystems. *IACR Cryptology ePrint Archive*, 2013:866, 2013.
29. R. Shahid, M. U. Sharif, M. Rogawski, and K. Gaj. Use of embedded FPGA resources in implementations of 14 round 2 SHA-3 candidates. In R. Tessier, editor, *FPT*, pages 1–9. IEEE, 2011.
30. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134, Santa Fe, New Mexico, Nov. 20–22, 1994. IEEE Computer Society Press.
31. S. Sinha Roy, F. Vercauteren, and I. Verbauwhede. High precision discrete Gaussian sampling on FPGAs. *Selected Areas in Cryptography, SAC 2013, Burnaby, British Columbia, Canada, August 14-16, to appear*, 2013. `http://www.cosic.esat.kuleuven.be/publications/article-2372.pdf`.
32. D. Suzuki and T. Matsumoto. How to maximize the potential of fpga-based dsps for modular exponentiation. *IEICE Transactions*, 94-A(1):211–222, 2011.
33. D. B. Thomas, W. Luk, P. H. W. Leong, and J. D. Villasenor. Gaussian random number generators. *ACM Comput. Surv.*, 39(4), 2007.
34. S. Vaudenay. Decorrelation: A theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, Sept. 2003.

A Appendix

A.1 Bernoulli Sampling

In this section we briefly recall the Bernoulli Sampling algorithms; for a detailed analysis we refer to [9].

The first tool for sampling in [9] is an algorithm to sample according to $\mathcal{B}_{\exp(-x/f)}$ for any positive integer x using $\log_2 x$ precomputed values as described in Algorithm 1.

Algorithm 1 Sampling $\mathcal{B}_{\exp(-x/f)}$ for $x \in [0, 2^\ell)$

Input: $x \in [0, 2^\ell)$ an integer in binary form $x = x_{\ell-1} \cdots x_0$
Precomputation: $c_i = \exp(-2^i/f)$ for $0 \le i \le \ell - 1$
  for $i = \ell - 1$ **to** 0
    if $x_i = 1$ **then**
      sample $A_i \leftarrow \mathcal{B}_{c_i}$
      if $A_i = 0$ **then** return 0
  return 1

Next, this algorithm is used to transform a simple Gaussian distribution to discrete Gaussian of arbitrary parameter σ. This simple Gaussian (called the binary Gaussian because the probability of each x is proportional to 2^{-x^2}) has parameter $\sigma_2 = \sqrt{1/(2\ln 2)} \approx 0.849$. It is straightforward to apply (Alg. 2) because of the form of its (unnormalized) cumulative distribution

Algorithm 2 Sampling $D_{\mathbb{Z}^+,\sigma_2}$

Output: An integer $x \in \mathbb{Z}^+$ according to $D^+_{\sigma_2}$
  Generate a bit $b \leftarrow \mathcal{B}_{1/2}$
  if $b = 0$ **then** return 0
  for $i = 1$ **to** ∞ **do**
    draw random bits $b_1 \dots b_k$ for $k = 2i - 1$
    if $b_1 \dots b_{k-1} \neq 0 \dots 0$ **then** restart
    if $b_k = 0$ **then** return i
  end for

Algorithm 3 Sampling $D_{\mathbb{Z}^+,k\sigma_2}$ for $k \in \mathbb{Z}$

Input: An integer $k \in \mathbb{Z}$ ($\sigma = k\sigma_2$)
Output: An integer $z \in \mathbb{Z}^+$ according to D^+_{σ}
  sample $x \in \mathbb{Z}$ according to $D^+_{\sigma_2}$
  sample $y \in \mathbb{Z}$ uniformly in $\{0, \dots, k-1\}$
  $z \leftarrow kx + y$
  sample $b \leftarrow \mathcal{B}_{\exp(-y(y+2kx)/(2\sigma^2))}$
  if $\neg b$ **then** restart
  return z

Algorithm 4 Sampling $D_{\mathbb{Z},k\sigma_2}$ for $k \in \mathbb{Z}$

Generate an integer $z \leftarrow D^+_{k\sigma_2}$
if $z = 0$ restart with probability $1/2$
Generate a bit $b \leftarrow \mathcal{B}_{1/2}$ and return $(-1)^b z$

$$\rho_{\sigma_2}(\{0,\dots,j\}) = \sum_{i=0}^{j} 2^{-i^2} = 1\,.\,1\,0\,0\,1\underbrace{0\dots0}_{4}1\underbrace{0\dots0}_{6}1 \quad \dots \quad \underbrace{0\dots0}_{2(j-2)}1\underbrace{0\dots0}_{2(j-1)}1\,.$$

From there, one easily builds the distribution $k \cdot D_{\mathbb{Z}^+,\sigma_2} + \mathcal{U}(\{0 \dots k-1\})$ as an approximation of $D_{\mathbb{Z}^+,k\sigma_2}$ which is corrected using rejection sampling technique (Alg. 3). This rejection only requires variables of the form $\mathcal{B}_{\exp(-x/f)}$ for integers x. The last step is to extend the distribution from $\mathbb{Z}^+$ to the whole set of integers $\mathbb{Z}$ as done by Algorithm 4.

Final Rejection Step with $\mathcal{B}_{1/\cosh(X)}$: To avoid explicit computation of $1/\cosh$ for the final rejection step, the authors of [9] suggested the following algorithm: $\mathcal{B}_{1/\cosh(X)} = \mathcal{B}_{\exp(-|X|)} \oslash \big(\mathcal{B}_{1/2} \vee \mathcal{B}_{\exp(-|X|)}\big)$. It is shown that it requires at most 3 calls to $\mathcal{B}_{\exp(-|X|)}$ on average.

Algorithm 5 Sampling $\mathcal{B}_a \oslash \mathcal{B}_b$

sample $A \leftarrow \mathcal{B}_a$
if A **then** return 1
sample $B \leftarrow \mathcal{B}_b$
if $\neg B$ **then** return 0
restart

A.2 Complements on the KL-Divergence

The following important properties and proofs apply to the KL-divergence.

Lemma 3 (Additivity of Kullbach-Leibler Divergence). *Let $\mathcal{P}_0, \mathcal{P}_1, \mathcal{Q}_0, \mathcal{P}_1$ be independant distributions over some countable set Ω. Then*

$$D_{KL}(\mathcal{P}_0 \times \mathcal{P}_1 \| \mathcal{Q}_0 \times \mathcal{Q}_1) = D_{KL}(\mathcal{P}_0 \| \mathcal{Q}_0) + D_{KL}(\mathcal{P}_1 \| \mathcal{Q}_1)$$

Lemma 4 (Non-increasing of Kullbach-Leibler Divergence). *Let $\mathcal{P}, \mathcal{Q}$ be independant distributions over some countable set Ω. Then for any function f*

$$D_{KL}(f(\mathcal{P})\|f(\mathcal{Q})) \leq D_{KL}(\mathcal{P}\|\mathcal{Q}).$$

Equality holds when f is injective over the support of $\mathcal{P}$.

Lemma (restatement of Lemma 2)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distribution of same support S. Assume that for some $\delta(i) \in (0, 1/4)$ we have a relative error bound: forall $i \in S, |\mathcal{P}(i) - \mathcal{Q}(i)| \leq \delta(i)\mathcal{P}(i)$. Then*

$$D_{KL}(\mathcal{P}\|\mathcal{Q}) \leq 2\sum_{i\in S}\delta(i)^2\mathcal{P}(i)$$

Proof. Once again we rely on second order Taylor bounds. For any $y > 0$, we have

$$\begin{aligned}\frac{d}{dx}y\ln\frac{y}{y+x} &= \frac{-y}{x+y} &&= -1 \text{ at } x = 0\\ \frac{d^2}{dx^2}y\ln\frac{y}{y+x} &= \frac{y}{(x+y)^2} &&\leq \frac{2}{y} \text{ if } |x| \leq \delta y\end{aligned}$$

Therefore we conclude that for any x such that $|x| \leq \delta|y|$,

$$|y\ln\frac{y}{y+x} + x| \leq \frac{2}{y}\delta^2y^2 = 2y\delta^2.$$

One now sets $y = \mathcal{P}(i)$ and $x = \mathcal{Q}(i) - \mathcal{P}(i)$ and sums over $i \in S$

$$|\sum_{i\in S}P(i)\ln\frac{\mathcal{P}(i)}{\mathcal{Q}(i)} + (\mathcal{Q}(i) - \mathcal{P}(i))| \leq 2\delta^2\sum_{i\in S}\mathcal{P}(i).$$

Since S is the support of both $\mathcal{P}$ and $\mathcal{Q}$, we have $\sum_{i\in S}\mathcal{P}(i) = \sum_{i\in S}\mathcal{Q}(i) = 1$, therefore we conclude that

$$\sum_{i\in S}P(i)\ln\frac{\mathcal{P}(i)}{\mathcal{Q}(i)} \leq 2\sum_{i\in S}\delta(i)^2\mathcal{P}(i).$$

□

Lemma (restatement of Lemma 1)**.** *Let $\mathcal{E}^{\mathcal{P}}$ being an algorithm making at most q queries an oracle sampling from a distribution $\mathcal{P}$ and outputting a bit. Let $\epsilon \geq 0$, and $\mathcal{Q}$ be a distribution $D_{KL}(\mathcal{P}\|\mathcal{Q}) \leq \epsilon$. Let x (resp. y) denote the probability that $\mathcal{E}^{\mathcal{P}}$ (resp. $\mathcal{E}^{\mathcal{Q}}$) outputs 1. Then,*

$$|x - y| \leq \frac{1}{2}\sqrt{q\epsilon}.$$

Proof. Assume that $x \geq y$ and $x \geq q\epsilon$ (the other cases makes the conclusion trivial). By the additive and non-increasing properties of the Kullback-Leibler divergence, we have

$$D_{\mathrm{KL}}(\mathcal{B}_{p'}, \mathcal{B}_p) = D_{\mathrm{KL}}\left(\mathcal{E}^{\mathcal{P}}\|\mathcal{E}^{\mathcal{Q}}\right) \leq D_{\mathrm{KL}}\left(\mathcal{P}^q\|\mathcal{Q}^q\right) \leq q\epsilon.$$

We conclude using Taylor bounds; from the identities

$$\frac{d}{dz} D_{\mathrm{KL}}(\mathcal{B}_z, \mathcal{B}_y) = \ln\left(\frac{z}{y}\right) - \ln\left(\frac{1-z}{1-y}\right) = 0 \quad \text{at } z = y$$
$$\frac{d^2}{dz^2} D_{\mathrm{KL}}(\mathcal{B}_z, \mathcal{B}_y) = \frac{1}{z(1-z)} \geq 4 \quad \text{when } z \in (0,1)$$

we obtain that $4(x-y)^2 \leq D_{\mathrm{KL}}(\mathcal{B}_x, \mathcal{B}_y) \leq q\epsilon$. □