Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on Falcon.

-author's version-

Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Ingrid Verbauwhede

imec-COSIC, KU Leuven
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract Sampling from discrete Gaussian distribution has applications in lattice-based post-quantum cryptography. Several efficient solutions have been proposed in the recent years. However, making a Gaussian sampler secure against timing attacks turned out to be a challenging research problem. In this work, we observed an important property of the input random bit strings that generate samples in Knuth-Yao sampling. We delineate a generic step-by-step method to instantiate a discrete Gaussian sampler of arbitrary standard deviation and precision by efficiently minimizing the Boolean expressions by exploiting this property. Discrete Gaussian samplers generated in this method can be up to 37% faster than the state of the art method. Finally, we show that the signing algorithm of post-quantum signature scheme Falcon using our constant-time sampler is at most 33% slower than the fastest non-constant time sampler.

1 Introduction

Hard lattice problems have been used to construct public-key cryptosystems since Ajtai's [2] seminal work in 1996. Shortly after, the discovery of NTRU [20] and Learning with errors (LWE) [29] further established lattice-based cryptography as a viable alternative to the popular RSA or elliptic curve based public-key cryptography. In the past decade, the looming threat of quantum computers and widely believed resistance of lattice problems against quantum computers has given the research in lattice-based cryptography a major thrust. Among these constructions the cryptosystems based on LWE (or its' ring variant RLWE [24]) has emerged to be most popular mostly due to their simple operations, security evaluation and strong worst case to average case reduction. Therefore, it is not a coincidence that in the NIST's recent standardization call for post-quantum cryptographic protocols [1], a majority of the submitted protocols are based on LWE or RLWE.

These problems usually require an *error* term to hide its secret keys. Traditionally, these error terms are sampled from a Gaussian distribution which itself is a non-trivial task. A lot of effort [26,16,14,9,17,32] have been devoted into generating these Gaussian samples efficiently. However, in almost all these methods the sampling process runs in non-constant time which opens up new avenues of side-channel attacks. Unfortunately, except for a few simple countermeasures [7,31], it was not known how to generate Gaussian samples that can resist these attacks. So the current trend is to generate samples from distributions e.g the binomial distribution [4,8] or the uniform random distribution [11,12,22] where it is easy to generate samples in constant-time and model those distributions as Gaussian distribution during security evaluation of the cryptosystem. This works well with encryption and key-exchange schemes but for signature schemes [3,5,15] this leads to larger key-sizes and costlier computations. The work in [21] introduced constant-time discrete Gaussian sampling by evaluating Boolean expressions. The efficiency of this sampling method was further improved by using bit-slicing. In the current work, we look back at this method to further improve the speed of this sampling algorithm.

2 Our Contribution

Our contributions in this paper can be summarized as below,

1. The work in [21] established the concept of constant-time discrete Gaussian sampling by evaluating Boolean expressions. However, that work did not provide concrete details of generating these Boolean functions. Here, we provide a detailed description of how to create these Boolean functions that maps random bits to the samples of an arbitrary discrete Gaussian distribution.
2. By carefully analyzing the the Knuth-Yao sampling for discrete Gaussian we observed a special property of the input random strings that generate samples. We show how we can take the leverage of this property to minimize the Boolean functions efficiently. This minimization technique can speed up the sampling by up to 37% compared to the simple minimization technique in [21].
3. Finally, we show that using Gaussian samples in a post-quantum signature algorithm can be both very practical and secure. We use Falcon signature algorithm to show that the performance of the algorithm does not degrade much with respect to the fastest non constant-time sampler and stays within practical limit even after we replace the non constant-time sampler with our constant-time sampler.

Organization of the paper: In Sec. 3, we provide some preliminaries which are useful to understand the rest of the paper. Sec. 4, briefly describes the constant-time sampler introduced in [21]. Sec. 5, contains description of our Boolean expression generation and efficient minimization techniques. Sec. 6 provides performance comparison of Falcon signature scheme with non constant-time and constant-time samplers. We draw conclusion in Sec. 7.

3 Preliminaries

In this section, we define different notations which we are going to use throughout this work. Later, we also present a brief description of the discrete Gaussian distribution and Knuth-Yao sampling which is used to generate samples from discrete Gaussian distributions. We define $\mathbb{Z}^* = \{0\} \cup \mathbb{Z}^+$. All the binary strings are evaluated in the reverse order, *i.e* binary evaluation of n-bit string $b = b_{n-1}b_{n-2}\cdots b_1b_0$ with b_0 being the LSB, is $2^{n-1}\cdot b_0 + 2^{n-2}\cdot b_1 + \cdots + 2\cdot b_{n-2} + b_{n-1}$. Boolean `and, or,` and `not` operations are denoted by the symbols &, |, ¯respectively. The terms Boolean function and Boolean expression are used interchangeably. We use f_η to denote Boolean functions that maps η Boolean variables to a single Boolean variable. If b is a Boolean variable then we denote repetition of b, i times $\underbrace{b\dots b}_{\text{i}}$ by b^i. Also, we denote a Boolean string of length i where each variable is either 0 or 1 by $(0/1)^i$. For the binary trees, we denote the level where children of the root exist as the 0-th level, children of these nodes reside at 1-st level and so on. Hence, starting from root we need $i+1$ *steps* to reach nodes at level i. We assume that the standard deviations in our sampler are *small* and our sampler can be used as a base sampler in [25,28] where samples from discrete Gaussian distribution with large standard deviation are generated by combining samples from discrete Gaussian distribution with small standard deviation. We also use σ to denote the standard deviation of a Gaussian distribution.

3.1 Discrete Gaussian distribution

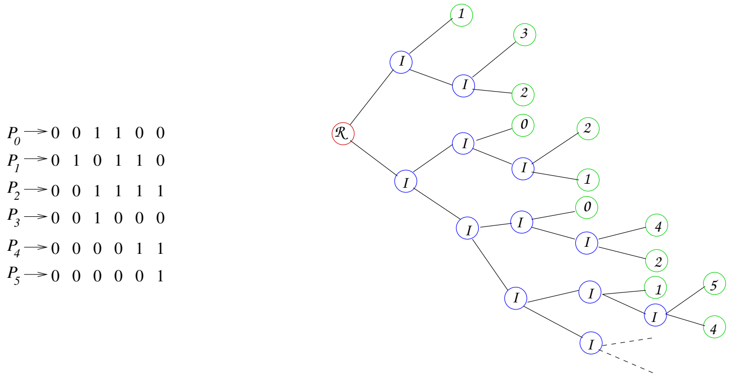
The probability distribution function of a discrete Gaussian distribution is given as,

$$\mathscr{D}_\sigma(X = z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(z-c)^2/2\sigma^2}.$$

Here, X is a random variable defined over $\mathbb{Z}$. In this work, we always consider discrete Gaussian distributions that are centered around 0 *i.e* $c = 0$. Due to the symmetry of probability density function it is sufficient to generate samples over $\mathbb{Z}^+$ and use a random bit to determine the sign. For most practical scenario, all the samples are generated in the interval $[0, \tau\sigma]$, where τ is a positive constant known as *tail-cut* factor. Also, again for practical reasons the probabilities $\mathscr{D}_\sigma(x)$ are calculated only up to n-bit precision, we denote this as $\mathscr{D}^n_\sigma(x)$.

3.2 Knuth-Yao Sampling

Dwarakanath and Galbraith [17] first discoursed on the idea of using the well known Knuth-Yao [23] sampling method to generate samples from discrete Gaussian distributions. This method can be divided into two stages. In the precomputation stage, for a particular standard deviation this method first creates a probability matrix of dimension $(\tau\sigma + 1) \times n$ where a row consists of $\mathscr{D}^n_\sigma(v)$ if



$P_0 \rightarrow 0\ 0\ 1\ 1\ 0\ 0$

$P_1 \rightarrow 0\ 1\ 0\ 1\ 1\ 0$

$P_2 \rightarrow 0\ 0\ 1\ 1\ 1\ 1$

$P_3 \rightarrow 0\ 0\ 1\ 0\ 0\ 0$

$P_4 \rightarrow 0\ 0\ 0\ 0\ 1\ 1$

$P_5 \rightarrow 0\ 0\ 0\ 0\ 0\ 1$

Figure 1: Probability matrix and corresponding DDG tree for $\sigma = 2$ and $n = 6$. Red, blue and green nodes denote the root, intermediate and leaf nodes respectively.

$v = 0$ and $2 \cdot \mathscr{D}_\sigma^n(v)$ for all other $v \in [1, \tau\sigma]$. Using this probability matrix a binary tree named discrete distribution generation (DDG) tree is created such that, hamming weight of i-th column of the probability matrix equals the number of leaf nodes in the i-th level of the tree and each leaf node contains a sample value in the sample space $[0, \tau\sigma]$. An example is shown in Fig. 1. During sampling, a random walk is started from the root node using random bits at every step to decide between the bottom or top subtree. The sampling stops when this random walk hits a leaf node and the sample value associated with the leaf node is returned as sample. It is worthwhile to note here that for a distribution D with finite support s.t $\sum_{x \in Supp(\mathrm{D})} \mathrm{P}^n(x) = 1$ where $\mathrm{P}^n(x)$ is the probability of x up to n-bit floating point precision under the distribution D, the DDG tree is finite and it is possible to generate samples that follow the distribution D *exactly*. Whereas, for distributions like discrete Gaussian distributions with infinite support the DDG tree grows infinitely and it is not possible to generate samples that follow the discrete Gaussian distribution *exactly*. In this case, the τ and n is chosen such that the statistical distance between the generated distribution and the actual distribution is lower than $2^{-\lambda}$ where λ is a security parameter.

In the following section, we describe an algorithm to generate samples from discrete Gaussian distribution using Knuth-Yao sampling efficiently.

3.3 Column scanning Knuth-Yao sampling

The column scanning Knuth-Yao sampling algorithm from [32] generates the DDG tree on-the-fly, thus making the sampling process time and memory efficient. This is shown in Alg. 1. We denote the Hamming weight of column i of the probability matrix P as $\mathrm{h_i}$. Define, $\mathrm{GAP^i}$ as,

$$\mathrm{GAP^i} = (b_0 \cdot 2^i + b_1 \cdot 2^{i-1} + \cdots + b_i) - (\mathrm{h_0} \cdot 2^i + \mathrm{h_1} \cdot 2^{i-1} + \cdots + \mathrm{h_i}) \qquad (1)$$

Algorithm 1: Knuth-Yao column scanning Sampling

```
input  : Probability matrix P
output: Sample value s
d ← 0;// Distance between the visited and the rightmost internal node
Hit ← 0;// 1 when sampling process hits a terminal node
col ← 0;// column number of probability matrix
while $Hit=0$ do
    $r \leftarrow RandomBit()$;
    $d \leftarrow 2 * d + r$;
    for $row=MAXROW$ down to 0 do
        $d \leftarrow d - P[row][col]$;
        if $d=-1$ then
            $s \leftarrow row$;
            $Hit \leftarrow 1$;
            $ExitForLoop()$;
    $col \leftarrow col + 1$;
return s
```

here b_j is the output of *RandomBit()* during the j-th iteration of the **while** loop in Alg. 1. It is evident from the above algorithm that a sample is found in the i-th column if and only if $\mathrm{GAP^i} < 0$ and $\mathrm{GAP^{i'}} \geq 0, 0 \leq i' < i$

4 Previous work

For quite some time, there have been efforts to mitigate the side channel information leakage from Gaussian samplers but they either sacrifice efficiency [7] for constant-time operation or do not provide adequate security [31]. The bit-sliced discrete Gaussian sampler in [21] proposed an efficient and constant-time Gaussian sampler. The key observation in that work was that there exists an unique path from root to each leaf of the DDG tree since it is a binary tree. This path is determined by the input random bits to the sampler. This implies that there exists a many-to-one mapping between the set of input random bit strings $(b_0 b_1 \cdots b_{n-1})$ to the set of sample bits. Further, this mapping can be expressed as a set of Boolean functions $f_n^{\iota},\ \iota \in [0, m-1]$, here m is the maximum possible bit length of any sample. This is shown in Fig. 2. Now, each sample bit can be calculated in constant-time by executing the corresponding Boolean function completely. The above method for sampling, though being a constant-time algorithm, has very poor efficiency. So, the next key observation was that modern processors has wide word length and bit wise Boolean operators which can be exploited to generate multiple sample in a batch in a single instruction multiple data (SIMD) fashion. In this method, assuming word length w, a variable $b_i^{var}, i \in [0, n-1]$ is packed with w input random bits $b_i^0, b_i^1, \cdots, b_i^{w-1}$. These random variables are then used to evaluate Boolean function f_n^{ι} using bitwise Boolean operators to generate variables $s_{\iota}^{var}, \iota \in [0, m, -1]$ which are packed

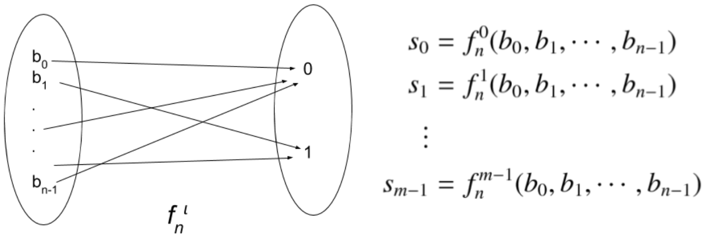

b_0
b_1
b_{n-1}
0
1
f_n^ι
$s_0 = f_n^0(b_0, b_1, \cdots, b_{n-1})$
$s_1 = f_n^1(b_0, b_1, \cdots, b_{n-1})$
$\vdots$
$s_{m-1} = f_n^{m-1}(b_0, b_1, \cdots, b_{n-1})$

Figure 2: Mapping n random bits to output sample bits and corresponding Boolean functions.

with w output sample bits $s_\iota^0, b_\iota^1, \cdots, b_\iota^{w-1}$. These variables are then unpacked to generate w samples at a time. Despite the overhead of packing and unpacking bits this method was shown to be approximately two times faster($\sigma = 6.15543$) than the best known alternative for constant-time sampling *i.e* cumulative distribution table(CDT) based sampling with linear search [7].

5 Our work

In this section, we first prove the Theorem 1, which establishes an important property of the structure of the input random bit strings which generate the samples. We then describe our Boolean minimization strategy for discrete Gaussian sampling. In this section, we also consider that the Knuth-Yao sampler always takes input random bit strings of length n. However, it is possible that a sample is found on the c-th level of the DDG tree or equivalently the sampler needs only $c+1$ bits to generate the sample. The remaining $n-(c+1)$ bits do not influence the outcome of the sampling, we call them *don't care*(x) bits.

Theorem 1 *All the random bit strings which generate samples are of the form* $x^i(0/1)^j01^k$, *where* $i, j, k \in \mathbb{Z}^*$, $i + j + k + 1 = n$ *and* x *is the* don't care *bits.*

Proof. It is enough to show that there is no random bit string of the form $x^i1^{k'}, k' \in \mathbb{Z}^*$ which generates a sample, *i.e* the sampling process do not hit any leaf node when the input random bit string is $1^{k'}$. We prove this by contradiction, let's assume that the bit string $1^{k'}$ hits a leaf node at $k'-1$-th level of the DDG tree. So, $\mathrm{GAP}^{\mathrm{k}'} < 0$. Since the random bit string is $1^{k'}$ this implies that for all $2^{k'}$ different input random bit strings $\mathrm{GAP}^{\mathrm{k}'} < 0$. In other words, all the random input bit strings generate some sample or hit some leaf nodes of the DDG tree. So, the DDG tree is a finite tree as the tree beyond $k'-1$-th level is not reachable by the Knuth-Yao sampling anymore and becomes redundant. Hence, by the property of Knuth-Yao sampling we can say that the DDG tree up to level $k'-1$ is able to generate samples *exactly* from the discrete Gaussian distribution of the given standard deviation. Or, in other words if the samples in the DDG tree up to $k' - 1$-th level lies in the interval $[0, \tau'\sigma]$, then

$$\sum_{x=-\tau\sigma}^{\tau'\sigma} \mathscr{D}_\sigma^n(x) = 1$$

But, this is impossible as discrete Gaussian distribution has infinite tail and the probabilities being real numbers have infinite precision, so the above summation will never be equal to 1. Hence, our assumption that the input bit string $x^i 1^{k'}, k' \in \mathbb{Z}^*$ generates a sample is therefore wrong. □

Also, experimentally we have seen that j is bounded by a *small* Δ i.e $j_{\max} \leq \Delta$. For example for $\sigma = 1, 2, 6.15543$ and 215 the Δ is $4, 4, 6$, and 15 respectively. In the next sections, we will show that by using these two facts we can develop a very efficient minimization technique for a fast constant-time Gaussian sampler.

5.1 Efficient minimization

To instantiate an efficient discrete Gaussian sampler with a specific standard deviation, we first enumerate the number of leaves in the DDG tree and the random bit strings that hit those leaf nodes. We create a list L of these random bit strings $x^i(0/1)^j 01^k$ with their corresponding binary decomposed sample values. It is evident form Sec. 3.2 that the size of this list is $\sum_{i=0}^{n-1} \mathrm{h_i}$.

Now, using this list L, it is possible to generate the Boolean functions f^ι that simply maps the input random bit strings to the sample bits. To improve the efficiency, we can use synthesis tools to minimize these Boolean expressions, since minimizing these Boolean expressions is equivalent to the well known circuit minimization problem which is a NP-complete problem. Also, as the number of variable n is large, these tools can only use heuristic minimization algorithms which can generate number of complications. Firstly, the behaviour of these heuristic algorithms is very unpredictable which may cause the resulting minimized expressions or in turn the discrete Gaussian sampler behave in a very unexpected manner, secondly minimization is not very efficient as the tools can use only heuristic algorithms, and finally most of these efficient heuristic algorithms are intellectual property of their respective corporations and the output of these algorithms cannot be put in the public domain, rendering them unusable for discrete Gaussian samplers for lattice based cryptography implementations which are mostly open source.

To overcome these problems we propose an alternate but efficient minimization strategy, we first sort the input random bit strings $x^i(0/1)^j 01^k$ and their corresponding sample bits in the list L in the ascending order of k. This makes all the input random bit strings with equal number of consecutive 1's from the LSB become adjacent in the list L this is shown in Fig. 3 for a discrete Gaussian sampler with $\sigma = 2$. In the next step, we divide the list L in sublists $\mathrm{l}_0, \mathrm{l}_1, \cdots \mathrm{l}_{\mathrm{n}'}$ such that all the input random bit string in the sublist l_κ has $\kappa \in [0, n']$ consecutive 1's from the LSB or is of the form $x^i(0/1)^j 01^\kappa$. As in the sublist l_κ, the $\kappa + 1$ least significant bits are fixed, the output sample bits in this sublist

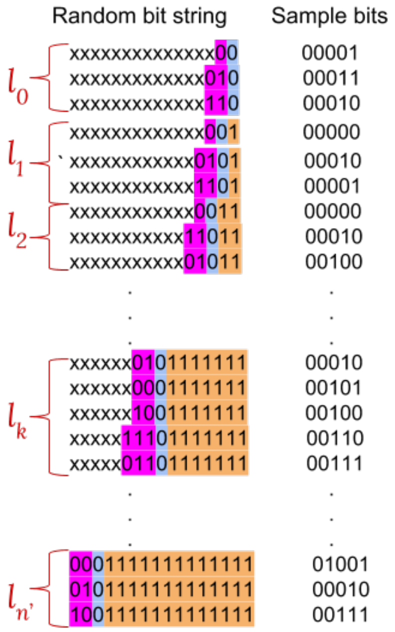


Figure 3: Dividing a List L in sublists l_κ for $n = 16$ and $\sigma = 2$. The rightmost bit is the LSB in both the columns.

are determined by the next j random bits only. Now, we recall that $j_{\max} \leq \Delta$. Hence, we can generate Boolean functions f^{ι}_{Δ}, that maps Δ input random bits to the output sample bits for each sublist. Since, Δ is *small*, we can now use very efficient minimization techniques to minimize the Boolean expressions f^{ι}_{Δ}. It is even practically feasible to use Karnaugh map or brute force techniques to minimize the expressions. In this work we used the open source tool `Espresso` with `-Dso -S1` options for *exact* minimization of each expression. We generate these Boolean expressions $f^{\iota,t}_{\Delta}, t \in [0, n']$ for all the sublists $\mathrm{l}_0, \mathrm{l}_1, \cdots \mathrm{l}_{\mathrm{n}'}$ and for all $\iota \in [0, m-1]$. In the next section, we will discuss how we can join these Boolean expressions together to create a constant-time discrete Gaussian sampler.

5.2 Constant-time sampling

We first recall a method to execute a non constant-time *if-else* block $\nu = \alpha \ ? \ \beta_0 : \beta_1$ in constant-time as $\nu = (\alpha \& \beta_0)|(\bar{\alpha} \& \beta_1)$ where α, β_i are binary variables. It is easy to extend this to a long *if-elseif-...-else* block as $\nu = (\alpha_0 \& \beta_0)|(\bar{\alpha}_0 \& ((\alpha_1 \& \beta_1)|(\bar{\alpha}_1 \& (\ldots|(\bar{\alpha}_n \& \beta_{n+1})))))$. Such methods for constant-time execution of *if-else* blocks have been known since as early as constant-time bit

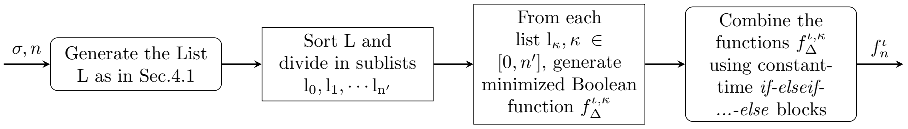

σ, n
Generate the List L as in Sec.4.1
Sort L and divide in sublists $\mathrm{l}_0, \mathrm{l}_1, \cdots \mathrm{l}_{n'}$
From each list $\mathrm{l}_\kappa, \kappa \in [0, n']$, generate minimized Boolean function $f_\Delta^{\iota,\kappa}$
Combine the functions $f_\Delta^{\iota,\kappa}$ using constant-time *if-elseif-...-else* blocks
f_n^ι

Figure 4: Flowchart efficient minimization of Boolean expressions f^ι for constant-time discrete Gaussian sampling.

sliced implementation of DES [6] and their constant-time behaviour have been studied well in the literature.

Now, consider the binary variable $c_\kappa = b_0 \& b_1 \& \ldots \& b_{\kappa-1} \& \bar{b}_\kappa$ where b_i's are the input random bits to the Gaussian sampler. Also, we recall from the previous section that the input random bits of sublist l_κ are of the form $x^i(0/1)^j 01^\kappa$. Now, we make the following claim,

Claim 1 *c_κ equals 1 if and only if the random bit string belongs to sublist l_κ.*

The above claim can be proven easily by using the structure of input random bits of list l_κ and the definition of the variable c_κ. We leave this proof to the reader. Using, Claim 1 and the constant-time *if-elseif-...-else* blocks as discussed in the beginning of this section we can combine the Boolean expressions $f_\Delta^{\iota,t}, t \in [0, n']$ to create a constant time Gaussian sampler with precision n as shown in Eqn. 2,

$$f_n^\iota = c_0 \ ? \ f_\Delta^{\iota,0} : (c_1 \ ? \ f_\Delta^{\iota,1} : (\ldots : (c_{n'-1} \ ? \ f_\Delta^{\iota,n'-1} : f_\Delta^{\iota,n'}))) \tag{2}$$

So far, we used only binary variables $\alpha, \beta, c_\kappa, b$ etc., to describe our methods. All of these methods can be trivially transformed into the bit- sliced SIMD setting as described in Sec.3.2 of [21] by using variables of larger bit lengths instead of binary variables and replacing single bit Boolean operators to their bit wise counterparts. We assert that our construction of the Boolean expression f_n^ι runs in constant-time as long as each of the Boolean expressions $f_\Delta^{\iota,t}$ runs in constant-time. Now, each of $f_\Delta^{\iota,t}$ is Boolean expression that computes ι-th bit of a sample from Δ random bits. Constant-time behaviour of such functions has been proven and analyzed rigorously in [21]. Moreover, we used the tool `"dudect"` described in [30] to affirm the constant running time of our algorithm. The whole process for efficient minimization of f_n^ι is shown as a flowchart in Fig. 4. We will provide a tool that implements the strategies mentioned here.[1] Fig. 5 shows histogram plots constant-time discrete Gaussian sampling of $\sigma = 2$ and 6.15543 using the methods described above.

6 Results and application to Falcon signatures

In this section, we provide performance results of our constant-time discrete Gaussian sampler. Since the lattice based signature scheme Falcon [18] uses

[1] Tool and the code available at `https://github.com/Angshumank/const_gauss_split`

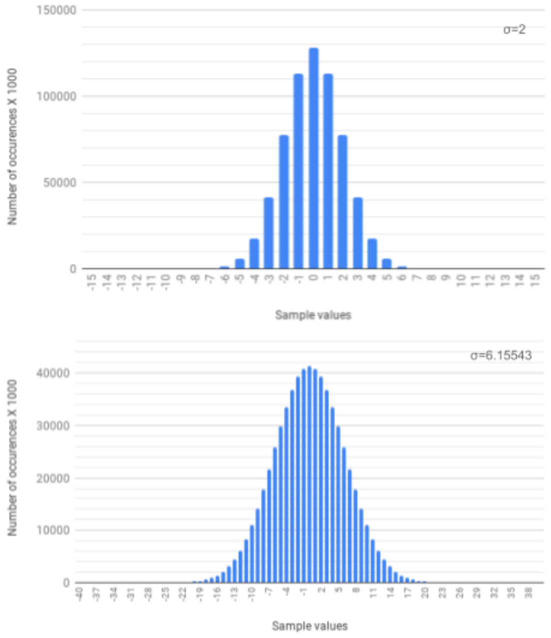


Figure 5: Histogram plot for $\sigma = 2$ and $\sigma = 6.15543$ using 64×10^7 samples.

discrete Gaussian samples during signing, we chose this scheme as our target application. Before describing our results later in this section, we give below a small description of the Falcon signature and the necessity of constant-time Gaussian sampling in the scheme.

Falcon signature scheme : The post-quantum signature scheme Falcon [18] has been submitted to NIST's [10] ongoing effort to standardize post-quantum protocols. It is a well known fact that post-quantum schemes are most often faster than their pre-quantum RSA or discrete logarithm based schemes but they tend to have a larger keys or signatures. As noted by the authors, the design of Falcon focuses on minimizing the combined bitsize of the public key and the signature. Indeed, among the lattice based signature schemes submitted to the NIST's standardization procedure Falcon has the smallest combined bitsize of public key and signature, this is partly due to their choice of Gaussian distribution and NTRU lattices. During the signing of messages Falcon requires samples from a discrete Gaussian sampler with σ low enough to produce optimal short vectors as required by the security of the scheme but no too low to leak secret information about the basis of the lattice. Depending on the number field used this σ can be either 2 or $\sqrt{5}$. In our work, we only used the considered the instance of Falcon with $\sigma = 2$, the other instance can be realized using the same methods described above. For more details about the Falcon signature scheme we refer the interested readers to the detailed documentation [18].

The Falcon signature scheme is a relatively newer scheme and robustness against side channel attacks of this scheme has not been analyzed rigorously like older lattice based signature schemes [14]. Though at this moment we are not aware of any attacks like [19,27] on Falcon which exploits non constant-time Gaussian sampler, the authors of Falcon have specifically mentioned that use of non constant-time Gaussian sampler during signing can be potentially harmful to the security of the scheme and serious effort should be dedicated for implementing constant-time Gaussian sampler.

For comparison, we plugged in our constant-time Gaussian sampler in the implementation [18] of Falcon provided by the authors and compare our results with two of their fastest non constant-time samplers, CDT sampler [26] and the byte-scanning CDT sampler [13]. Additionally, we also provided a comparison with the linear search based constant-time CDT sampler [7] with our sampler. The precision $n(= 128)$, the tail-cut $\tau(= 13)$ and the pseudo-random number generator have been kept same for all the instances. We compiled all implementations using `gcc-5.4` with flags `-O3 -fomit-frame-pointer -march=native -std=c99`. The computation times are measured on a single core of a Intel(R) Core(TM) i7-6600U processor running at 2.60GHz and disabling hyper-threading, Turbo-Boost, and multi-core support as standard practice on Ubuntu 16.04 running on a Dell Latitude E7470 laptop. As our target processor has 64-bits our sampler can generate 64 samples in a batch. The results are shown in Table. 1

Security level	Non constant-time (Signs/Sec)		Constant-time (Signs/Sec)	
	Byte-scanning CDT	CDT	Linear search CDT	This Work
Level 1 (N=256)	10327	8041	6080	7025
Level 2 (N=512	5220	4064	3027	3527
Level 3 (N=1024)	2640	2014	1519	1754

Table 1: Comparing performances of Falcon-sign with non constant-time and constant-time samplers with `ChaCha` as the pseudo random number generator. N is a security parameter which refers to the degree of quotient polynomial used to define the number field used in Falcon.

We can see from Table. 1 that replacing the our constant-time sampler with the fastest non constant-time sampler reduces the performance of signing algorithm by approximately 32% at worst, whereas replacing our sampler with non constant-time CDT sampler slows down the signing algorithm by at most 13%. Also, the signing algorithm with our sampler is at least 15% faster than the linear search based CDT sampling. It is very important to note that the table based methods such as byte-scanning CDT, CDT or linear search CDT also gets advantage of the caching in our target Intel processors for fast table search due to the small size of the tables. Overall, these results show that removing side-channel vulnerabilities of non constant-time discrete Gaussian sampler with our constant-time Gaussian sampler does not hamper the performance of signing algorithm to a large extent.

In Table 2, we compare the performance of the Gaussian sampler with our efficient minimization technique and the Gaussian sampler with simple minimization described in [21]. We can see that for $\sigma = 2$ we get around 37% improvement, but the improvement for $\sigma = 6.15543$ is approximately 11%, the reason behind this is that for $\sigma = 6.15543$ in [21] the output of the minimization tool has been manually optimized further for efficiency.

	Constant-time sampler in [21]	This work	Improvement
$\sigma = 2$	3,787	2,293	37%
$\sigma = 6.15543$	11,136	9,880	11%

Table 2: Comparing discrete Gaussian sampler with our efficient minimization with the previous described in [21]. The numbers in the table are in clockcycles and do not include the overhead for generating the pseudorandom numbers.

7 Conclusion

We can see that the sampling from Gaussian distributions can be efficient using the methods described in [21] and this work. But there is a fixed cost of generating pseudorandom numbers that makes the whole sampling process slower. As mentioned in [21], we have also noticed in this work that $80 - 85\%$ of the total time is spent on generating the pseudorandom numbers. It is possible to reduce this overhead by using some different method for generating the pseudorandom numbers. For example, if used `ChaCha` stream ciphers (as in the Falcon reference implementation) instead of `Keccak` to generate pseudorandom numbers this overhead reduces to approximately 60%. It is also possible to use more platform specific alternatives like AES-NI instructions on Intel processors to reduce this overhead. Also, a good research direction is to develop statistical measures like Rényi [28] divergences or max-log [25] distances to reduce the precision requirement of discrete Gaussian sampling and hence reducing the requirement of pseudorandom numbers. One of the reason the centered binomial distribution [4] samplers are efficient is that they need very few pseudorandom bits to generate a sample. Our experiments suggest that improvements in the direction of reducing the requirement of pseudorandom bits per sample and efficient generation of these bits combined with the methods described here can make the discrete Gaussian sampling very competitive.

8 Acknowledgements

This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement Cathedral ERC Advanced Grant 695305 and by EU H2020 project FENTEC (Grant No. 780108) and by the Hercules Foundation AKUL/11/19.

References

1. Nist post-quantum cryptography round 1 submissions. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions (2017), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions, [Online; accessed 12-April-2018]
2. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing. pp. 99–108. STOC '96, ACM, New York, NY, USA (1996), http://doi.acm.org/10.1145/237814.237838
3. Akleylek, S., Bindel, N., Buchmann, J., Krämer, J., Marson, G.A.: An efficient lattice-based signature scheme with provably secure instantiation. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings, pp. 44–60. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-31517-1_3
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange – a new hope. In: USENIX Security 2016 (2016)
5. Barreto, P.S.L.M., Longa, P., Naehrig, M., Ricardini, J.E., Zanon, G.: Sharper ring-lwe signatures. Cryptology ePrint Archive, Report 2016/1026 (2016), https://eprint.iacr.org/2016/1026
6. Biham, E.: A fast new des implementation in software. In: Biham, E. (ed.) Fast Software Encryption. pp. 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
7. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy. pp. 553–570 (May 2015)
8. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634 (2017), http://eprint.iacr.org/2017/634
9. Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., Weiden, P.: Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. In: Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282. pp. 402–417. Springer-Verlag New York, Inc., New York, NY, USA (2014), http://dx.doi.org/10.1007/978-3-662-43414-7_20
10. Chen, L., Jordan, S.P., Liu, Y.K., Moody, D., Peralta, R.C., Perlner, R.A., Smith-Tone, D.C.: Report on post-quantum cryptography. In: NIST Internal Report (NISTIR) - 8105 (2016), http://dx.doi.org/10.6028/NIST.IR.8105

11. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, Report 2016/1126 (2016), http://eprint.iacr.org/2016/1126
12. D'Anvers, J.P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2018. pp. 282–305. Springer International Publishing, Cham (2018)
13. Du, C., Bai, G.: Towards efficient discrete gaussian sampling for lattice-based cryptography. In: 2015 25th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–6 (Sept 2015)
14. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, pp. 40–56. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40041-4_3
15. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems 2018(1), 238–268 (Feb 2018), https://tches.iacr.org/index.php/TCHES/article/view/839
16. Ducas, L., Nguyen, P.Q.: Faster gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings, pp. 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34961-4_26
17. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete gaussians for lattice-based cryptography on a constrained device. Applicable Algebra in Engineering, Communication and Computing 25(3), 159–180 (2014), http://dx.doi.org/10.1007/s00200-014-0218-3
18. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over ntru (2018), https://falcon-sign.info/, [Online; accessed 10-October-2018]
19. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, pp. 323–345. Springer Berlin Heidelberg, Berlin, Heidelberg (2016), http://dx.doi.org/10.1007/978-3-662-53140-2_16
20. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. pp. 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
21. Karmakar, A., Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Constant-time discrete gaussian sampling. IEEE Transactions on Computers 67(11), 1561–1571 (Nov 2018)
22. Karmakar, A., Bermudo Mera, J.M., Sinha Roy, S., Verbauwhede, I.: Saber on arm. cca-secure module lattice-based key encapsulation on arm. IACR Transactions on Cryptographic Hardware and Embedded Systems 2018(3), 243–266 (Aug 2018), https://tches.iacr.org/index.php/TCHES/article/view/7275

23. Knuth, D., Yao, A.: Algorithms and Complexity: New Directions and Recent Results, chap. The complexity of nonuniform random number generation. Academic Press (1976)
24. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings, pp. 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13190-5_1
25. Micciancio, D., Walter, M.: Gaussian sampling over the integers: Efficient, generic, constant-time. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II, pp. 455–485. Springer International Publishing, Cham (2017)
26. Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings, pp. 80–97. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14623-7_5
27. Pessl, P.: Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In: Dunkelman, O., Sanadhya, S.K. (eds.) Progress in Cryptology – INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings, pp. 153–170. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-49890-4_9
28. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, pp. 353–370. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-44709-3_20
29. Regev, O.: New lattice-based cryptographic constructions. vol. 51, pp. 899–942. ACM, New York, NY, USA (Nov 2004), http://doi.acm.org/10.1145/1039488.1039490
30. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: 2017 Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017. p. 14 (2017)
31. Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Compact and side channel resistant discrete gaussian sampling. Cryptology ePrint Archive, Report 2014/591 (2014), https://eprint.iacr.org/2014/591.pdf
32. Sinha Roy, S., Vercauteren, F., Verbauwhede, I.: High precision discrete gaussian sampling on fpgas. In: Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282. pp. 383–401. Springer-Verlag New York, Inc., New York, NY, USA (2014), http://dx.doi.org/10.1007/978-3-662-43414-7_19