# Constant-time Discrete Gaussian Sampling

**–authors' version–**

Angshuman Karmakar[1], Sujoy Sinha Roy[1], Oscar Reparaz[1], Frederik Vercauteren[1,2], and Ingrid Verbauwhede[1]

[1]KU Leuven ESAT/COSIC and Imec
Kasteelpark Arenberg 10 bus 2452, B-3001 Leuven-Heverlee, Belgium
[2]Open Security Research, Fangda 704, Kejinan-12th, Nanshan, 518000 Shenzhen, China
`firstname.lastname@esat.kuleuven.be`

**Abstract.** Sampling from a discrete Gaussian distribution is an indispensable part of lattice-based cryptography. Several recent works have shown that the timing leakage from a non-constant-time implementation of the discrete Gaussian sampling algorithm could be exploited to recover the secret. In this paper, we propose a constant-time implementation of the Knuth-Yao random walk algorithm for performing constant-time discrete Gaussian sampling. Since the random walk is dictated by a set of input random bits, we can express the generated sample as a function of the input random bits. Hence, our constant-time implementation expresses the unique mapping of the input random-bits to the output sample-bits as a Boolean expression of the random-bits. We use bit-slicing to generate multiple samples in batches and thus increase the throughput of our constant-time sampling manifold. We also show a method to relax the constraints of constant-time sampling negligibly to further increase the efficiency. Our experiments on an Intel® i5-Haswell processor show that our method can be as much as 5.5 times faster than the constant-time implementation of cumulative distribution table based sampling and much more memory efficient than the Knuth-Yao algorithm with shuffling for a similar level of security.

**Keywords:** Knuth-Yao, Constant-time sampling, Lattice-based cryptography

## 1 Introduction

Public-key cryptography (PKC) eliminated one serious drawback of otherwise highly efficient symmetric-key cryptography, namely key establishment among all the communicating parties or the requirement of a central key distribution authority. The security of such cryptosystems are assured by underlying computationally hard problems. Since the discovery of the Diffie-Hellman [DH76] key exchange protocol, the popularity and utility of PKC has grown steadily over the past few decades. Currently, primitives derived from RSA and ECC are used extensively for public-key cryptography on a wide range of devices. In comparison to symmetric-key cryptography, the major drawbacks of PKC are larger key sizes and slower running time. To get the best of both worlds, contemporary security protocols use both schemes in tandem for highly efficient and secure digital security solutions.

Unfortunately, large-scale quantum computers running Shor's [Sho97] and Proos-Zalka's [PZ03] algorithms can solve the underlying hard problems of RSA and ECC. In this scenario, lattice-based PKC [Reg04,Ajt96] has become an attractive choice to provide digital security in the post-quantum world. The confidence in security arises from the fact that unlike RSA and ECC, there is no known algorithm that can use quantum computers to solve the underlying hard problems of lattice-based cryptography. Hard lattice problems like Learning with errors (LWE) [Reg04] and Short Integer Solutions [Ajt96] and their ring equivalents R-LWE and R-SIS [LPR10,Mic07] are some of the prominent choices to build various lattice-based cryptography protocols. In fact, there exists a wide variety cryptography primitives that can be built on top of these problems. For example, Digital signature schemes [DDLL13,BLN$^+$16,ABB$^+$16], public-key encryption [LPR10,LP11], key-exchange protocols [ADPS16,BCD$^+$16], identity-based encryption [GPV08,CHKP10,ABB10a,ABB10b], fully homomorphic encryption [BV11,BGV14,Bra12,Gen09]. The other features which make lattice-based cryptography a suitable alternative is, proven worst case to average case reduction of lattice problems and somewhat simpler operations than other PKC primitives, namely discrete Gaussian sampling and matrix-vector or polynomial multiplication.

LWE is a system of *approximate linear equations* with the secret key being the solution of the system. LWE uses noise to hide its secret parameters without which the system can be easily solved using Gaussian elimination. This noise is typically sampled from a discrete Gaussian distribution. Sampling from such distribution involves either storing a large table of precomputed values or computing the exponential function to a very high precision. Hence, Gaussian sampling accounts for a non-negligible share of resources in a lattice-based cryptography implementation. For example, In the case of BLISS [DDLL13] and Lyubashevsky's [Lyu12,WHCB13] signature scheme, the Gaussian sampling alone takes about 35% and 50% of the total running time of the signature algorithms respectively. Since the beginning of lattice-based cryptography, a lot of research has been performed to reduce the storage and computational overhead of sampling [Pei10,DN12,DDLL13,BCG$^+$14,DG14,RVV14]. The discrete Gaussian sampler is arguably most vulnerable to side channel attacks in a lattice-based cryptography implementation. Currently, as lattice-based cryptography is becoming more efficient and being implemented in a wide variety of devices, it is imperative to make the sampling secure against side channel attacks. Different methods have been proposed to make the sampling efficient and resource friendly but there is a lack of research to make the Gaussian sampling secure against side-channel attacks. This was not a cause for a serious concern as there was no attack available that could efficiently exploit the side channel leakage information against the cryptosystem. Recently, Bruinderink et al. [GBHLY16] has described a very effective side channel attack on the BLISS digital signature scheme. They exploited the irregular cache memory access pattern of the Gaussian sampler. Moreover, the authors have also shown that all the presently known Gaussian samplers are vulnerable to their attack. This work was quickly followed by Peter Pessl [Pes16], who mounted this attack on the same scheme with shuffled samples that was proposed as a side channel security measure for Gaussian sampling by Roy et al. [RRVV14]. The latter attack requires much more samples (but still practical) than the previous one. Though there are some simple countermeasures like constant-time table scanning [BCNS15] or the previously mentioned shuffling method that can eliminate or mitigate the side channel leakage, they come with a performance cost of the sampling operation and do not scale very well for larger standard deviations. We also note that, due to the side channel vulnerability of discrete Gaussian sampling, currently there is a trend to design lattice-based cryptography schemes that do not use Gaussian sampling in the performance critical part of the scheme [BLN$^+$16,ABB$^+$16]. These schemes however require more arithmetic operations and larger modulus for security.

### 1.1 Our Contributions

In this paper, we describe a method to sample from a discrete Gaussian distribution securely. Our contributions can be summarized as follows.

- Almost all of the currently known efficient samplers use a table of precomputed values and binary search, which are the main sources of side channel leakage of such samplers. In this work, we avoid the use of tables. More precisely, we analyze the Knuth-Yao discrete Gaussian sampling [DG14] and observe a unique mapping between the output sample values and input random bits of the sampling algorithm. We utilize this observation to express the output sample values as a Boolean function of the input random bits. During sampling, each of these Boolean functions are evaluated in constant-time to generate each sample, thus making the sampling procedure a constant-time operation. This is described in Section 3.1.
- In Section 3.2, we show how we can exploit a bit-slicing methodology to generate samples in batches. This increases the throughput of our sampler by orders of magnitude. This enhancement in performance is achieved by carefully tweaking the way random input bits are stored and utilizing bitwise operators and the wide data path of modern processors.
- In Section 3.3, we illustrate a technique to increase the efficiency of our sampler by slightly sacrificing the rigidity of constant-time sampling. This trade-off decreases the latency to generate samples but leaks some information about the samples, namely the range of possible values within which the generated samples lie. We discuss the security implications of such leakage in Section 3.4.
- In Section 4, we compare our method to other secure discrete Gaussian samplers for a similar level of security. We provide an experimental comparison of run times using a C implementation on a Intel® i5-Haswell processor. Additionally, we describe a method to split a discrete Gaussian distribution with large standard deviation into many smaller discrete Gaussian distributions with smaller standard deviation.

- In Sections 4.2 and 4.2, we also provide results of our sampling method implemented on FPGA and with AVX vector instructions utilizing the wider data path.
- Finally in Section 5, we provide a side channel analysis of our sampling algorithm.

## 2 Discrete Gaussian Sampling

In this section, we provide a brief discussion on discrete Gaussian sampling and different methods to generate samples from such distribution.

### 2.1 Definition

The probability distribution function $D_{\mathbb{Z},\sigma}$ of a discrete Gaussian distribution defined over $\mathbb{Z}$ with mean $\mu = 0$ and standard deviation $\sigma$ is defined as,

$$D_{\mathbb{Z},\sigma}(X = z) = \frac{1}{S} e^{-z^2/2\sigma^2}.$$

Here $X$ is a random variable defined over $\mathbb{Z}$ and $S$ is the normalization constant, defined as,

$$S = \sum_{x=-\infty}^{\infty} e^{-x^2/2\sigma^2} \approx \sigma\sqrt{2\pi}.$$

To generate samples over $Z$, it is sufficient to generate samples over $\mathbb{Z}^+$ and use a single random bit to determine the sign due to the symmetry of discrete Gaussian distribution across its mean.

Ideally, the support of a Gaussian distribution has range $(-\infty, \infty)$, but in most practical applications, it is neither feasible nor required to generate samples from this range. Instead, *tail-cut* factor $\tau$ is used to generate samples from a smaller range $[-\tau\sigma, \tau\sigma]$, ignoring other values beyond this range that has a very low probability of occurrence. Also, as the probabilities of $D_{\mathbb{Z},\sigma}$ , $x \in [-\tau\sigma, \tau\sigma]$ are real numbers, their binary expansion can be infinitely long. In practice, the probabilities are calculated only up to a certain precision $\lambda$ depending upon the requirement of the application. For most lattice-based cryptography applications the values of $\tau$ and $\sigma$ are chosen as 12 and 128 respectively, such that the generated samples are statistically very close to the ideal Gaussian distribution. Traditionally, statistical distance was used to measure this closeness. But recently the work of Bai et al. [BLL$^+$15] has shown that the value of $\tau$ can be reduced to as low as 6 using the Rényi divergence as the closeness measure. It is worth noting here that for a Gaussian distribution decreasing $\tau$ also decreases the precision $\lambda$. Further Saarinen [Saa15] has proposed that reducing $\lambda$ to 64 does not harm the security. In this work we assume $\tau = 12$ and $\lambda = 128$ and note that our method can be trivially adapted for other values of $\tau$ and $\lambda$.

### 2.2 Sampling from a discrete Gaussian

Sampling from a continuous Gaussian distribution has a wide range of applications in different fields of natural science, social sciences, mathematics, and engineering. Hence, it has been studied extensively for long time. Sampling from a discrete Gaussian distribution is a comparatively less studied topic. Since the start of their use in lattice-base cryptography, several methods have been proposed to sample from a discrete Gaussian distribution. Some of them are rejection sampling [DN12], cumulative distribution table (CDT) based sampling [Pei10], discrete Ziggurat sampling [BCG$^+$14], Knuth-Yao sampling [DG14], and Bernoulli sampling [DDLL13]. Among these methods, the rejection sampling does not require any storage of precomputed tables but requires many random bits and many repetitions. Hence, it does not perform very well in practice. All other methods use precomputed tables and binary search for efficient sampling. Here we discuss CDT sampling and Knuth-Yao sampling as these two methods can be more efficiently [HKR$^+$16] instantiated as leakage-resistant sampling algorithm than others.

**The CDT based sampling:** The CDT based sampling precomputes a cumulative distribution function (CDF) table $T$ for $i \in [-\tau\sigma, \tau\sigma]$ according to the given discrete Gaussian distribution with $\lambda$ bits of precision, such that $T[i+1]-T[i] = D_\sigma(i)$. The sampling phase of the algorithm is basically a search operation on the CDF table $T$. First, a random $r \in [0,1)$ is generated then the table $T$ is searched to find an $s$, such that $T[s+1] \geq r > T[s]$. If such an $s$ is found, it is returned as the sample. To reduce the storage requirement for the sampling, only the interval $[0, \tau\sigma]$ needs to be searched, as explained in Section 2.1. To improve the efficiency, binary search or improved versions of binary search such as binary search with guide table [PDG14,DB15] are used. In this method, the irregular table access pattern of binary search makes the sampling process vulnerable to cache-timing attacks which was used by Bruinderink et al. [GBHLY16].

**The Knuth-Yao sampling :** The Knuth-Yao [KY76] sampling algorithm was proposed to generate samples from any source of known probability distribution. The sampling algorithm uses a rooted binary tree which in this context is also known as a discrete distribution generation (DDG) tree. The DDG tree is constructed from the probability matrix, which is a matrix constructed from the samples in the support of the distribution and their corresponding binary expanded probabilities up to a certain precision. The probability matrix and the DDG tree are related as follows: the number of leaf nodes in the DDG tree at $i^{th}$ level is equal to the Hamming weight of the $i^{th}$ column of the probability matrix. Each leaf node of the DDG tree corresponds to a sample in the sample space. An example of the probability matrix and the corresponding DDG tree is shown in Fig. 1 for an arbitrary distribution with a sample space $S$ consisting of four samples.

$$\begin{array}{cccc c}
p_0 & 0 & 1 & 1 & 0 \\
p_1 & 0 & 1 & 0 & 1 \\
p_2 & 0 & 0 & 1 & 1 \\
p_3 & 0 & 0 & 1 & 0
\end{array}$$

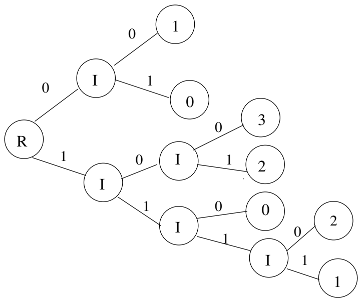


**Fig. 1.** A Probability matrix and the DDG tree corresponding to it. The random bits $\{0, 1\}$ are used to traverse the tree starting from the root.

The sampling operation is a random walk on the DDG tree. The random walk starts from the root and at each non-leaf node a random bit is generated to determine the direction of the random walk in the left or right sub-tree. The random walk stops when it hits a leaf node and the corresponding sample is returned. Here, the non-constant running time and branching during the random walk expose the cache vulnerability of the sampling operation.

Dwarakanath and Galbraith [DG14] first adapted the Knuth-Yao algorithm to sample from a discrete Gaussian distributions. Their work was later extended by Roy et al. [RVV14] with a more simplistic design methodology and reduced memory requirement. We refer the interested readers to their work for further details.

### 2.3 Previous works

As noted in Section 1, there has not been much research on the construction of constant-time Gaussian samplers, largely because of non-existence of efficient attacks. However, the existing non constant-time Gaussian samplers can be used for secure Gaussian sampling by applying some simple countermeasures. In this section we briefly revisit them.

**Constant-time table access** The table based Gaussian samplers use binary search for efficiency, which also makes them vulnerable to timing attacks. These algorithms can be converted to secure sampling algorithms by replacing the binary search with constant-time linear search of the whole table. This removes the cache-weakness of the binary search. But this countermeasure does not scale very well, as it may be an acceptable option for Gaussian distributions with smaller standard deviations but for larger standard deviations linear search of the whole table to generate each sample incurs a significant overhead. Bos et al. [BCNS15] used this method for a leakage-resistant Gaussian sampling in their key-exchange scheme.

**Shuffling** Constant-time table access for Knuth-Yao sampling is more complicated and inefficient than the other table-based sampling methods. Roy et al. [RRVV14] proposed a method to mitigate the problem of side-channel leakage of the Knuth-Yao sampler using extra memory. Their method caches the first $k$ columns of the probability matrix in a table with $2^k$ entries. The table entries are either a sample value or an intermediate position in the DDG tree. The sampling operation of this algorithm can be divided in a secure and a non-secure part. In the secure part, the algorithm generates a $k$ bit random index and looks for the entry in the table. If the entry is a sample value, then it is returned. In the non-secure part, if the table entry holds a position in the DDG tree, then a random walk is commenced from that position to find a sample. In this scenario, the algorithm leaks the absolute values of the samples due to the difference in timing to find a sample. As a second countermeasure, the authors suggest a random permutation of the leaked and non-leaked samples after the sampling to obfuscate the locations of the samples from the attacker. Also, as the security of this method depends on the number of columns cached, the memory requirement of this procedure increases exponentially with an increase in the levels of security.

**Fixed step binary search** In their work, Howe et al. [HKR$^+$16] proposed a fixed step binary search for secure Gaussian sampling. In their proposal, the binary search always runs for $O(\log(n))$ steps where $n$ is the size of the table, irrespective of whether the sample has been found in a previous step or not. While this method may work for some specific platform, it is not a generic solution for constant-time sampling. The binary search will leak secrets on a wide variety of platforms.

## 3 Constant-time Knuth-Yao sampling

In this section, we analyze the Knuth-Yao sampling algorithm. We describe our observation on correlation between samples and input random bit-strings. Based on this observation we propose a constant-time Knuth-Yao discrete Gaussian sampling. We also propose two optimization schemes to increase the throughput of our sampling algorithm. We conclude the section with a security discussion of our sampler.

**Choice of sampling algorithm** At this point, we describe our rationale for choosing the Knuth-Yao sampling algorithm for constant-time sampling. During our initial investigation for a constant-time Gaussian sampler, we found two possible methods to devise a constant-time sampler. One is the simple constant-time linear table search, the other method is to somehow express each sample as a function of input bit-string and then execute the function in constant-time for each sample. The former method is a well known method and has been used before. But, there is no precedence in the literature of the latter method for constant-time sampling. For the second method we require a well defined mapping from the input random bit-string to the output samples. We found that due to the random-walk algorithm of Knuth-Yao sampling it is easier to find such a mapping (explained later) and hence a function that can be executed efficiently to find samples from the input random bit-string. We also stress that we do not claim that such an efficient function cannot be derived from other sampling algorithms. Further research in this field may yield such efficient functions from other sampling algorithms too.

Other reasons for choosing the Knuth-Yao algorithm are efficiency and low entropy consumption. The Knuth-Yao and CDT (or its variants) are very popular choices for implementation of lattice-based cryptography schemes due to their very efficient performance across different platforms. Howe at al. [HKR$^+$16] has also recommended Knuth-Yao and CDT for constant-time Gaussian sampling.

### 3.1 Our observation: mapping random bits to samples

As explained in Section 2.2, the Knuth-Yao sampling is a random walk that starts from the root of a DDG tree until it hits a terminal node (Fig. 1). At each node, a random bit is used to select the sub-tree which will be explored next. Hence, the path from the root of the tree to each terminal node is determined by a unique bit string. As each terminal node corresponds to a sample in the sample space, there exists a mapping from the set of random bit strings to the sample space.

Clearly, this mapping is many-to-one. For example in Fig. 1, sample 0 is returned when the bit string is 01 or 110. Or, if the random bits are extracted from random bit strings of length 4 then sample 0 is returned when the bit string is $01xx$ or $110x$, where $x$ can be either 0 or 1. Using the above observation, we can formulate the samples or the bits of the samples ($s_i$) as a binary function of the random bit strings ($r = r_0 \cdots r_{n-1}$) as Eq. (1), assuming the samples can have maximum $m$ bits and the probability matrix has $n$ columns.

$$\begin{aligned} s_0 &= f^0(r_0, r_1, \cdots, r_{n-1}) \\ s_1 &= f^1(r_0, r_1, \cdots, r_{n-1}) \\ &\vdots \\ s_{m-1} &= f^{m-1}(r_0, r_1, \cdots, r_{n-1}) \end{aligned} \tag{1}$$

To calculate a bit $s_i$ of a sample, the respective binary expression $f^i$ applies the corresponding set of binary operators on the input random bit string ($r_0 r_1 \cdots r_{n-1}$) irrespective of its value. Hence, for any $i^{th}$ bit $s_i$ of the sample, the computation time $t_i$ is always the same for any random input bit-string $r$. As an illustration, the arbitrary distribution given in Fig. 1, the sample space has only 4 samples, hence $m = 2$ and $n = 4$. The bits of the samples ($s_0$, $s_1$) can be calculated as,

$$\begin{aligned} s_0 &= \bar{r_3} r_2 \bar{r_1} r_0 \ \vee \ \bar{r_3} \bar{r_2} r_1 \bar{r_0} \\ s_1 &= \bar{r_3} \bar{r_2} r_1 \end{aligned}$$

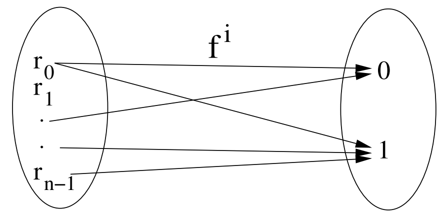


**Fig. 2.** Mapping $f^i : \{0,1\}^n \rightarrow \{0,1\}$ from set of random bit strings to the bits of samples

It is worth noting that to ensure constant-time sampling, all the binary operators should be applied in the order specified by Eq. (1) for each sample regardless of the input random bit string. We use a bit-slicing methodology and bit-wise operators for this purpose. This will be explained in Section 3.2.

Each sample can thus be generated in *constant-time* by computing each of its bits in constant-time. Unlike other Gaussian sampling methods, this method neither requires a large precomputed table nor an expensive computation such as computation of exponential functions with high precision. However, this method requires a *larger program memory* to store the formulae $f^i$.

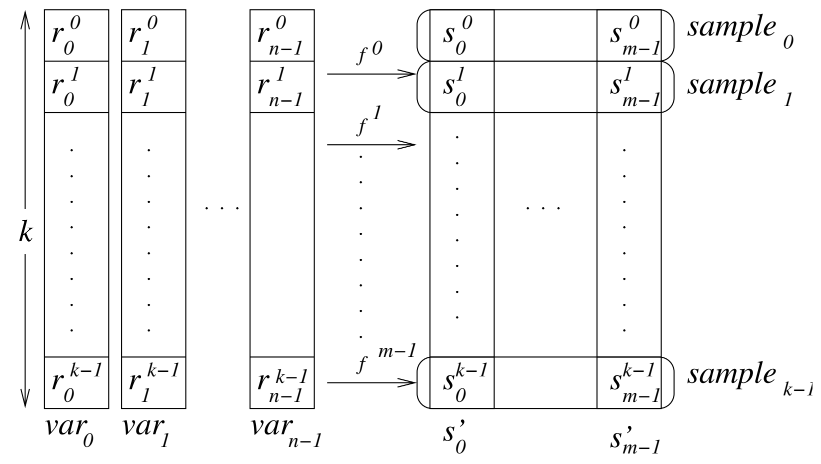
## 3.2 Batching the sampling process

Bit-slicing is a Single Instruction Multiple Data (SIMD) operation to improve the efficiency of a programme by exploiting data level parallelism. Starting from Biham's implementation of DES [Bih97], cryptographers have been using this method to speed up the execution of their algorithms for long time. Also, implementations using bit-slicing offers some immunity against side-channel attacks. Earlier, this method has been used for a fast and side-channel secure AES implementations [KS09,RSD06].

In this section, we describe a method to speed up our sampling by generating multiple samples at a time using bit-slicing. We utilize bit-wise Boolean operators on full processor words to achieve this. As shown in Eq. (1), each sample bit can be written as a function of $n$ random bits. In the simplest approach, we can store the $n$ random bits in $\lceil n/w \rceil$ variables, where $w$ is the word length of the processor and compute the sample bits by extracting the random bits from variables as required. This way of generating sample bits is very inefficient and has a very low throughput.

However, using an efficient storage of random bits we can greatly improve the throughput. Let's assume, we want to generate $k$ samples. So according to Eq. (1), we need $nk$ bits. To store these bits efficiently such that we can use bit-slicing, we take $n$ variables each of which stores $k$ bits. The bit $j \in [0, k-1]$ of the variable $i \in [0, n-1]$ represents the input random bit $r_i$ for sample $j$ as in Eq. (1). In other words, the $i^{th}$ variable stores all the $i^{th}$ input random bits to generate $k$ samples (Fig. 3). We can then apply the bit-wise operators on these variables as indicated by Eq. (1). Alternatively, we can rewrite Eq. (1) as :

$$
\begin{aligned}
s'_0 &= f^0(var_0, var_1, \cdots, var_{n-1}) \\
s'_1 &= f^1(var_0, var_1, \cdots, var_{n-1}) \\
&\vdots \\
s'_{m-1} &= f^{m-1}(var_0, var_1, \cdots, var_{n-1})
\end{aligned}
\tag{2}
$$



$k$
$r_0^0$ $r_0^1$ $r_0^{k-1}$
$r_1^0$ $r_1^1$ $r_1^{k-1}$
$r_{n-1}^0$ $r_{n-1}^1$ $r_{n-1}^{k-1}$
$var_0$ $var_1$ $var_{n-1}$
$f^0$ $f^1$ $f^{m-1}$
$s_0^0$ $s_0^1$ $s_0^{k-1}$
$s_{m-1}^0$ $s_{m-1}^1$ $s_{m-1}^{k-1}$
$s'_0$ $s'_{m-1}$
$sample_0$
$sample_1$
$sample_{k-1}$

**Fig. 3.** Efficient storage of random bits and sample bits for bit-slicing. Here $r_i^j$ represents $i^{th}$ input random bit of $j^{th}$ sample. Similarly, $s_i^j$ represents $i^{th}$ output bit of $j^{th}$ sample.

Where each variable $s'_t$ contains the $t^{th}$ sample bits $s_t,\ t \in [0, m-1]$ of $k$ samples. These variables are then used to extract the output sample bits to construct $k$ samples. Here, evidently the maximum value of $k$ is the word size $w$.

Therefore, using bit-wise Boolean operations and efficiently organizing the storage of input random bits, we can generate $w$ samples simultaneously. This is explained in Fig. 3.

## 3.3 Optimization: Dividing the input bit string

In this section, we describe a trade-off that leaks some information about the samples in favour of decreasing the latency. This method can be applied on top of the methods already described in Sections 3.1 and 3.2 for more efficient sampling. This optimization is based on the fact that the random walk on the DDG terminates within the first few levels of the tree with a very high probability. In other words, the probability of the random walk terminating at a level of the tree decreases rapidly with an increase in the level. This is explained below.

Let the Hamming weight of column $i$ of the probability matrix be $h_i$ and $P_i$ the probability that Knuth-Yao sampling algorithm finds a sample within the $i^{th}$ level of the DDG tree.

$$P_i = h_0 \cdot 2^{-1} + h_1 \cdot 2^{-2} + \cdots + h_i \cdot 2^{i+1} \tag{3}$$

Then the probability of not finding a sample within the $i^{th}$ level of the DDG tree is $P'_i = P'_{i-1} - h_i/2^{i+1}$. Fig. 4 shows this for the Gaussian distribution with $\sigma \approx 6.15543$ (refer to Section 4.1). It is evident that the probability $P'_i$ decreases exponentially with an increase in the the level of the DDG tree. For example, from Fig. 4 we see that the probability of not finding a sample within level 32 is $\approx 2^{-28}$. Hence we can say that most of the samples are found within the first 32 levels with a very high probability.

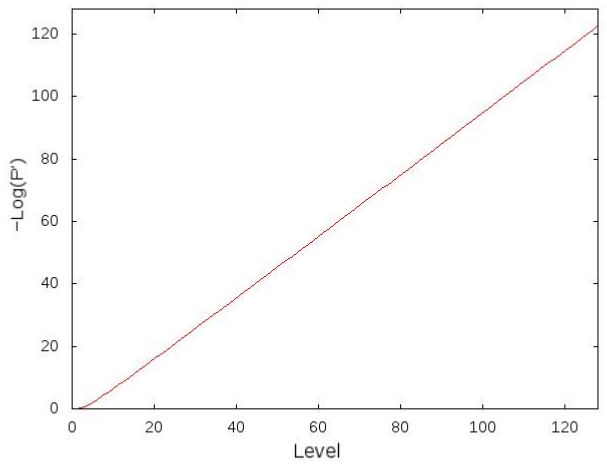


**Fig. 4.** $-\log_2(P'_i)$ vs $Level(i)$

We can use the above observation to make the sampling even more efficient. Let's define a function $\phi^i_\eta$, which takes a random input bit-string $r'$ of length $\eta$ and outputs the $i^{th}$ bit of a sample if the random walk corresponding to the bit-string $r'$ terminates within level $\eta$ of the DDG tree; else it returns null. To correlate with Eq. (1), $f^i = \phi^i_{128}$. We now divide the input random bit string in $k$ chunks of length $\delta_j$, $s.t \sum_0^{k-1} \delta_j = 128$. We also define functions $\phi^i_{\eta_0=\delta_0}, \phi^i_{\eta_1=\eta_0+\delta_1}, \cdots, \phi^i_{\eta_{k-1}=\eta_{k-2}+\delta_{k-1}}$. Finally, we can rewrite the functions $f^i$ in Eq. (1) by combining the functions $\phi^i_{\eta_j}$, $j \in [0, k-1]$ defined on the smaller chunks of input bit-string. During sampling, we calculate the functions $\phi^0_{\eta_j}, \phi^1_{\eta_j}, \cdots, \phi^{m-1}_{\eta_j}$ first before calculating functions $\phi^0_{\eta_{j+1}}, \phi^1_{\eta_{j+1}}, \cdots, \phi^{m-1}_{\eta_{j+1}}$. If the sample is found after calculating $\phi^0_{\eta_j}, \phi^1_{\eta_j}, \cdots, \phi^{m-1}_{\eta_j}$ we stop the sampling process and output the sample. If the chunks are big enough, we can say from our previous observation that evaluating only the first functions *i.e* $\phi^0_{\eta_0}, \phi^1_{\eta_0}, \cdots, \phi^{m-1}_{\eta_0}$ will be sufficient to find a sample with very high probability. We summarize our sampling algorithm in Algo. 1. It is evident that the if-else blocks in our sampling algorithm inherently introduce some information leakage. We discuss this in Section 3.4.

As mentioned before, for this Gaussian distribution $P'_{32} \approx 2^{-28}$, the program executes the $for\ loop$ in Algo. 1 more than once with probability $2^{-28}$, which makes it nearly a constant-time algorithm.

**Algorithm 1:** Discrete Gaussian sampling algorithm (with $k$ chunks $\delta_0, \delta_1, \cdots, \delta_{k-1}$)

```
  input  : Random bit string $r_0 r_1 \cdots r_{n-1}$
  output: Sample value s
1 for $i = 0$ to $k - 1$ do
2     Fetch next $\delta_i$ random bits;
3     Compute functions $\phi^0_{\eta_i}, \phi^1_{\eta_i}, \cdots, \phi^{m-1}_{\eta_i}$;
4     if none of the functions return null then
5         Compute $s$ from the sample bits;
6         return s
7     else
8         Continue;
9 return $FAIL$
```

To compare the efficiency gained from dividing the input bit string, we executed our sampler (Algo. 1) for different chunk sizes $\delta_0$. The results are given in Table 1.

| Chunk size $\delta_0$ | 32 | 64 | 96 | 128 |
|---|---|---|---|---|
| Probability of not finding a sample $P'$ | $2^{-28}$ | $2^{-59}$ | $2^{-91}$ | $2^{-122}$ |
| Clock-cycles (excluding random number generation) | 1649 | 3500 | 6190 | 10527 |

**Table 1.** Time to generate samples for different chunk size of input bit-string

### 3.4 Security

The optimization described in Section 3.3 turns the fully constant-time algorithnm to a *partially* constant-time algorithm. The if-else block in Algo. 1 reveals to the attacker a range of values $[0 - a]$, $a \in [0, \tau\sigma]$ where our generated sample lies. In our previous example of the Gaussian distribution with $\sigma \approx 6.15543$, if we divide the bit string in chunks of 32 bits *i.e* $\delta_0 = 32$, the smallest interval that the attacker can guess using timing information is $[0 - 39]$ *i.e* $[0 - 6.5\sigma]$ (this value depends on the DDG tree. In our case, the maximum sample value within 32 levels of the DDG tree is 39). This is in contrast with the shuffling method [RRVV14] which actually leaks the values of some of the samples and the leakage could be exploited to mount timing attacks [Pes16]. We argue that for such a sufficiently wide range of values this leaked information offers no particular advantage to the attacker. First of all, to the best of our knowledge there is no attack which will gain an advantage with the knowledge of a shorter range of samples except the exhaustive search and Arora-Ge [AG11] linearization attack. In both of these cases, for sufficiently wide interval (in our example $[0 - 6.5\sigma]$) computational complexity or the number of LWE samples required is higher than the best known algorithm to solve LWE. Moreover, Bai et al. [BLL$^+$15] showed that using Rényi divergence instead of the classical statistical distance as a security measure, sampling within a shorter interval instead of a large interval of $[0 - 12\sigma]$ does not harm the security. Here, we want to mention that our algorithm still outputs samples within $[0 - 12\sigma]$ but this is not always necessary. According to the requirements of the scheme, the width of the intervals can be increased or decreased by dividing the input random bit-string in smaller or bigger chunks respectively. Naturally, one can always revert the optimization from Section 3.3 to achieve a fully constant-time sampler.

## 4 Performance and comparison

In this section, we compare our method with the CDT based constant-time algorithm using a C implementation. For the performance measurement, we use a discrete Gaussian distribution with standard deviation $\sigma \approx 6.15543$. In the next section, we justify our choice of this standard deviation.

### 4.1 Splitting the Gaussian distribution

The BLISS-I [DDLL13] signature scheme uses a standard deviation $\sigma = 215$. However, as memory requirement to store the precomputed table increases with increase in $\sigma$, sampling from a Gaussian distribution with such a large standard deviation is difficult due to large memory requirement. Also, due to the large precomputed tables, generating samples securely is highly inefficient. Pöppelman et al. [PDG14] described a method to split this large standard deviation into two Gaussian distributions with smaller standard deviation and later combining them to create a distribution with large standard distribution. They used

| Algorithm | | Time (in clockcycles) | | | |
|---|---|---|---|---|---|
| Excluding random number generation | CDT sampling | $\lambda = 64$ | | $\lambda = 128$ | |
| | | 9363 | | 16060 | |
| | Our algorithm | $\delta_0 = 32$ | $\delta_0 = 64$ | $\delta_0 = 96$ | $\delta_0 = 128$ |
| | | 1649 | 3500 | 6190 | 10527 |
| Including random number generation (SHA-512) | CDT sampling | $\lambda = 64$ | | $\lambda = 128$ | |
| | | 22361 | | 38509 | |
| | Our algorithm | $\delta_0 = 32$ | $\delta_0 = 64$ | $\delta_0 = 96$ | $\delta_0 = 128$ |
| | | 6803 | 13530 | 22212 | 31065 |

**Table 2.** Comparison of clock cycles for different constant time sampling with similar probability of leakage for $\sigma \approx 6.15543$ to generate 64 samples on an 3.3 GHz intel i5-Haswell processor using only one core. The SHA-512 implementation from openssl 1.0.1e has been used for pseudo-random number generation.

Kullback-Liebler divergence, which is Rënyi divergence of order 1 [BLL$^+$15] instead of the more usual notion of statistical distance to show that the distribution created in this way is very *close* to the actual distribution. The formula to calculate KL divergence of a distribution $P$ from the target distribution $Q$ is shown below.

$$D_{KL}(P||Q) = \sum_{i \in S} \ln\big(\frac{P(i)}{Q(i)}\big) P(i) \tag{4}$$

We extend their work by splitting the Gaussian distribution further in 4 smaller distributions. We use Algo. 2 instead of a theoretical approach to maintain the desired divergence.

We discuss the method by Pöppelman et al. very briefly here. To generate a sample $x \leftarrow D_\sigma$, two samples $x_1, x_2 \leftarrow D_{\sigma_1}$ are generated, and combined as $x_1 + k_1 x_2$. The $\sigma, \sigma_1$ and $k_1$ are related as $\sigma_1 = \frac{\sigma}{\sqrt{1+k_1^2}}$, for $\sigma = 215$, $k_1 = 11$ and $\sigma_1 \approx 19.5$. The Kullback-Liebler divergence of the sampled data created in this way from the actual distribution is $\leq 2^{-128}$. We split the standard deviation one more level. We split $\sigma_1$ such that $\sigma_2 = \frac{\sigma_1}{\sqrt{1+k_2^2}}$. Consequently, to generate a sample $x \leftarrow D_\sigma$ we generate 4 samples $x_1,\ x_2,\ x_3,\ x_4\ \leftarrow D_{\sigma_2}$ and combine them as $x\ = (x_1 + k_2 x_2) + k_1(x_3 + k_2 x_4)$ .

Algo. 2 is used to calculate KL-divergence with two level splitting. After experimenting with different values of $\tau_1$ and $k_2$ we found that setting $\tau_1 = 14$ and $k_2 = 3$ produces $\sigma_2 \approx 6.15543$ which has the desired divergence from a Gaussian distribution with $\sigma = 215$.

**Algorithm 2:** Calculation of Kullback-Liebler divergence

```
1 divergence ← 0;
2 for $i = 0$ to $12 \cdot \sigma$ do
3     P(i) ← 0;
4     for all possible $(x_1, x_2, x_3, x_4) \in [0, \tau_1\sigma_2]$, s.t. $i = (x_1 + k_2 \cdot x_2) + k_1 \cdot (x_3 + k_2 \cdot x_4), [k_1 = 11, k_2 = 3]$ do
5         $P(i) \leftarrow P(i) + \prod_{j=1}^{4} D_{\sigma_2}(x_j)$;
6     $Q(i) = D_\sigma(i)$;
7     divergence ← divergence + $\ln\left(\frac{P(i)}{Q(i)}\right)$P(i);
8 return divergence
```

## 4.2 Performance

Our sampling algorithm takes 1649 clock cycles ($\delta_0 = 32$, for other values of $\delta_0$ refer to Table 2) to generate 64 samples from $\sigma \approx 6.15543$ on an Intel® i5-Haswell processor running CentOS. Per sample, 103 clock cycles are needed for $\sigma = 215$, used in BLISS-I [DDLL13]. If we include the cost to generate the pseudo-random numbers using SHA-512 from openssl 1.0.1e, it takes 6803 clock cycles to generate 64 samples with $\sigma = 32$ and 425 clock cycles to generate a single sample with $\sigma = 215$. Our high level implementation in C is only optimized by -O3 optimization of gcc. For efficiency, the Boolean functions $f^i$ in Section 3.1 used to generate samples should be minimized. We used the simple logic minimization tool ESPRESSO for this purpose[1].

As mentioned in Section 2.3, non-constant time methods can be converted to timing-attack resistant sampling methods using different countermeasures, which sacrifices their efficiency for security. Also, Howe et al. [HKR+16] has compared and analyzed such constant-time instantiations of different sampling algorithms. Their work shows that Knuth-Yao sampling with shuffle and constant-time cumulative distribution table (CDT) based methods are the most efficient for constant-time sampling. In this section, we compare our method with two of these methods for a similar probability of leakage.

The constant-time CDT sampler accesses all the elements of the CDF table for each sample. However, for a fair comparison with our method, instead of accessing the full table for each sample, we let the sampling method access each element for a part of the table and if the sample is not found the sampling method

[1] The code is available in `https://github.com/Angshumank/const_gauss`

searches in a bigger part of the table. For instance, in our previous example with $\sigma \approx 6.15543$ and $\delta_0 = 32$, initially we let the sampling algorithm search in an interval of $[0 - 39]$ or $[0 - 6.5\sigma]$. If the sample is not found in that interval, the sampling process searches in a bigger interval of $[0 - 56]$ or $[0 - 9\sigma]$ and so on. This method ensures the probability of leakage is similar to our method. Since the CDT method performs comparisons between the random string and the table entries, we use either 64-bit or 128-bit comparisons taking into account the 64-bit word length of the processor.

We implemented both methods in the C programming language and compiled with -O3 flag in gcc-4.8 on a CentOS desktop with intel core-i5-Haswell processor. The results are shown in Table 2.

The Knuth-Yao sampler with shuffling proposed by Roy et al. [RRVV14] is another method to prevent information leakage from the sampler. The method is described briefly in Section 2.3. The method caches the first $k$ columns of probability matrix in a table with $2^k$ entries. To compare it with our method for a similar probability of leakage with our sampler with $\delta_0 = 32$, we need $k = 32$ which requires $2^{32}$ memory and a massive overhead for linear searching the table. Moreover, Bruinderink et al. [GBHLY16] suggest that this method only increases the complexity of their atack. Peter Pessl [Pes16] exploited this weakness of the sampler to break the BLISS signature scheme with an increased number of signatures.

**Results using SIMD instructions** New generation of Intel® (starting with Sandy Bridge) and AMD® (starting with Bulldozer) provides support for Advanced Vector Instructions (AVX). These instructions are an extension of the x86 instruction set architecture and facilitates SIMD processing on data of width up to 128 bits. Later, starting with Haswell processors, Intel® introduced the AVX2 instruction set which increased this bit width to 256 bits. As described in Section 3.2, the throughput of our sampling algorithm can be readily increased by using AVX instruction sets. In Table 3, we report the time taken by our sampler to produce 256 samples at a time using AVX2 instructions. These results show almost 2x speed-up of our sampling algorithm using AVX2 instructions.

| Chunk size $\delta_0$ | 32 | 64 | 96 | 128 |
|---|---|---|---|---|
| Clock-cycles (excluding random number generation) | 2789 | 6478 | 11565 | 19605 |

**Table 3.** Time to generate 256 samples using AVX2 instructions.

**Results in hardware** To evaluate the performance of the proposed constant-time sampling algorithm in hardware, we designed the architecture of Fig. 5. We assume that the random bits are generated by an external source and received by the architecture in a serial fashion. The 112-bit register *random buffer* stores the input random bits. A counter is used to record the number of random bits received. After *random buffer* is filled with the input random bits (detected by the counter), the *enable* signal becomes true, and then the random bits are processed by the parallel combinational circuits *Block-0* to *Block-3*. These circuits implement the Boolean expressions of the sample-bits. In our architecture, *Block-0, Block1, Block2* and *Block-3* compute on the random bits of *random buffer* with the index ranges 0-31, 27-58, 54-86 and 80-111 respectively. All of these blocks compute in parallel and the output of the sampling operation is selected using the output-multiplexer.

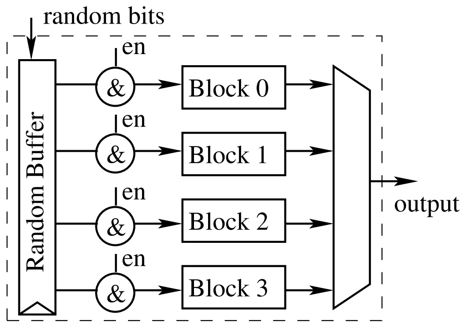


**Fig. 5.** Hardware architecture for constant-time sampling

Assuming an 8-bit port for random number input, it requires 14 cycles to fill *random buffer*. Only one cycle is spent by the parallel blocks to evaluate the Boolean expressions of the sample calculation. Hence, the architecture requires 15 cycles to finish one sampling operation.

We evaluated the architecture on a Xilinx Virtex-6 FPGA *xc6vcx75t-2ff484*. As per place-and-route report, the architecture consumes 997 slice registers and 2,682 slice LUTs and has a critical path delay of 4.9ns.

## 5 Evaluation

The implementation from Section 3 follows best-practice guidelines for constant-time code: constant program flow (no conditional branches), no secret-dependent memory accesses, and no usage of integer division nor multiplication operations. However, the fact that the high-level code looks constant time is no guarantee for the actual execution being constant time. Any piece in the tool chain may introduce a source of timing variability: in an extreme case, a very clever compiler would substitute the whole constant-time sampler with a faster, non constant-time one. Compilers and COTS architectures are currently designed to optimize for speed, code size, energy or power, but not security.

Thus, we resort to actual measurements to evaluate whether the resulting executable code runs in constant time on our platform or not. The evaluation of this section is empirical in nature and thus is bounded to the specific architecture, compiler and platform used.

## 5.1 Methodology

To assess timing variability we use leakage detection tests. Leakage detection tests were introduced by Coron, Naccache and Kocher [CKN00,CNK04] shortly after the introduction of DPA [KJJ99] and were targeted towards hardware side-channel evaluations. Nowadays, this technique has been proven to be useful also for timing variability evaluation. In this section, we follow the methodology and test code from [RBV17].

**Leakage detection for PRNGs** Generally speaking, we want to assess whether or not an adversary gets any advantage in distinguishing output samples from timing side-channel information. For that, we will deploy timing leakage detection tests to detect dependency between the execution time of the sampling procedure and input value to the Gaussian sampler. If the test fails to detect any dependency, the implementation is deemed secure. Note that the opposite outcome (there is detected leakage) is a necessary, but not sufficient, condition for an attack to work.

We design the timing leakage detection test as follows. We define two classes based on the input seed to the Gaussian sampler (that is, the input seed is treated as secret, and we aspire to detect any leakage dependent on this secret value). The two classes are defined as this: one class corresponds to a fix seed value; the other class is defined as a random seed value (fix-vs-random test). This choice, in contrast to a fix-vs-fix test, is expected to capture a broad set of leakages [DS16].

## 5.2 Platform

We perform the following experiments on the same platform from Section 4.2. We note that cycle counts are performed with the high-resolution Time Stamp Counter (RDTSC instruction).

## 5.3 Constant-time version

The first implementation is the constant-time variant of Section 3.2. This version does not early abort and is meant to be constant time by design. We carry the evaluation to confirm that this is actually the case, i.e., the compiler or any other micro-architectural components do not introduce any source of timing variability.

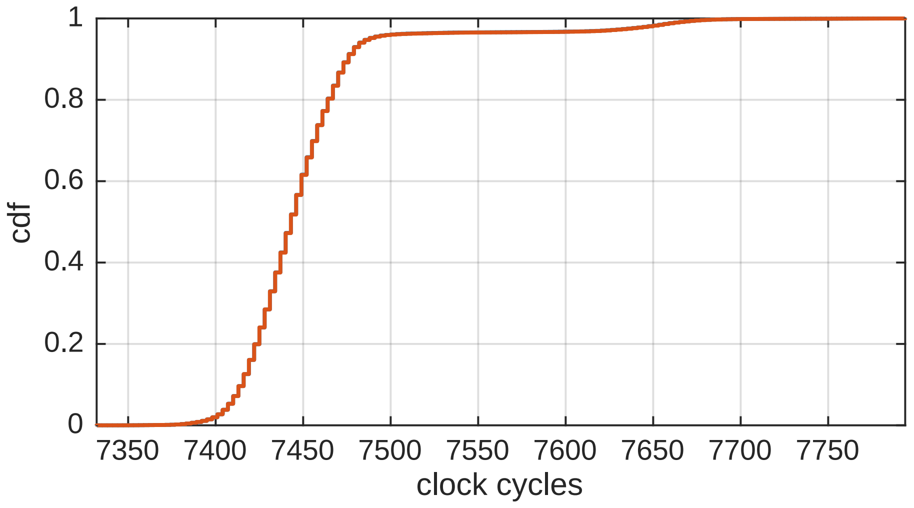


**Fig. 6.** Timing distribution cdfs for two classes in a fix-vs-random timing leakage detection test. Constant time sampler

In Fig. 6 we plot the empirical cumulative distribution functions for both timing distributions, corresponding to the two classes on input values (fix or random). The two distributions are actually indistinguishable and their cdfs overlap. We can see that the distributions are centered around 7450 cycles and there is a small class-independent variability ($\approx$ 100 cycles). This measurement noise could be caused by spurious interruptions by the operating system, or by the processor itself (for example, due to branch mis-predictions). The leakage detection t-test statistic does not surpass the threshold of $\pm 4.5$ and hence does not detect any leakage with up to 6 million iterations of the bitsliced sampling process. Various pre-processing options were explored with identical results.

We also perform a Kolmogorov-Smirnov (KS) test. The advantage is that it may detect that two distributions are different even if they share the same mean. The value of the statistic is 0.000625 which is lower than the cutoff value 0.001282. Thus, the KS test cannot reject the null hypothesis that both distributions are identical.

## 5.4 Early-abort sampler

We then proceed to test the optimized variant of Section 3.3. This version early-aborts if all samples are found within the first tree levels. The two cdfs are plotted in Fig. 7. The two distributions look indistinguishable, but the t-test can indeed separate both distributions. After 10 million measurements, the t-statistic takes value 330, well above the threshold $\pm 4.5$. Thus, the test clearly detects leakage.

We augment the study with a deeper investigation on the leakage source. The reasonable assumption is that the leakage comes from the early abort. We want

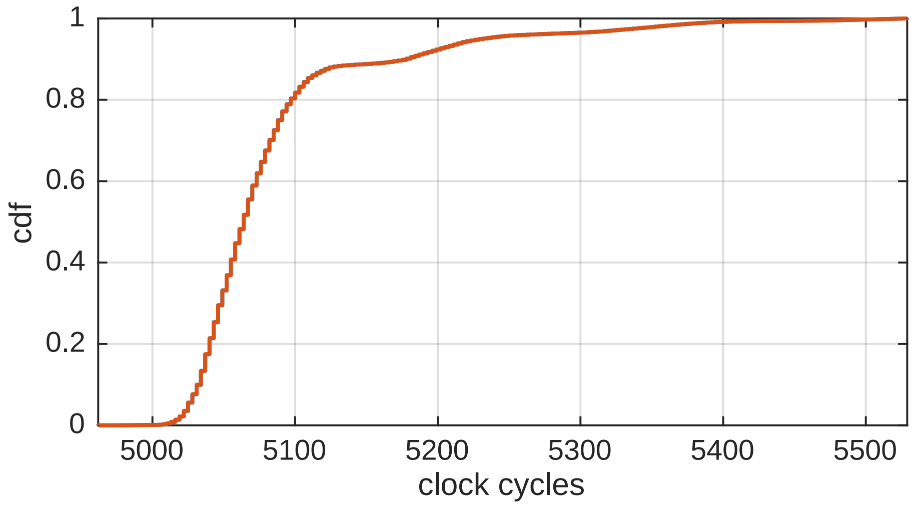


**Fig. 7.** Timing distribution cdfs (analogous to Fig. 6). Early-abort sampler

to ensure that the timing information leaked to the adversary is *only* whether the random walk terminated early or not. To test this, we devise two more experiments, aiming at decomposing the timing variability.

**Inter-stage variability** In this first test, we define the two classes as follows: one class contains random elements that always succeed in the first level of the tree (first stage); the other class contains random elements that always succeed in the second level (but not in the first). We expect to see severe leakage. In Fig. 9 we can see the cdfs of both distributions. The two classes are clearly separable. The t-statistic gives values around 600 with only 300 000 samples, indeed confirming the hypothesis of severe leakage.

**Intra-stage variability** The second test measures timing variability between inputs that are known to succeed in the first stage. That is, we craft input at random under the condition that the sampling will succeed in the first stage. Any leakage detected would indicate that not only whether or not the random walk terminated early leaks, but *also* some additional information on sample values is leaked. The expected behavior is naturally that no leakage is detected.

In Fig. 9 we plot the cdfs for both class-conditional distribution. The distributions are apparently indistinguishable. This is corroborated with a t-test. The t-statistic achieves a value of 2.7 after 100 million iterations, not surpassing the threshold of $\pm 4.5$ and hence not being able to disprove the null hypothesis of "leakage not present". The K-L test does not detect leakage either.

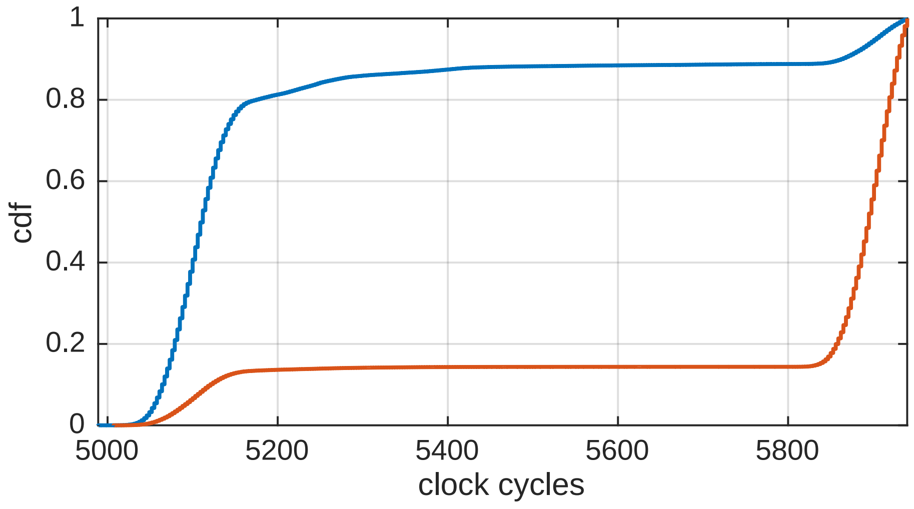


**Fig. 8.** Timing distribution cdfs. Early-abort sampler, inter-stage variability

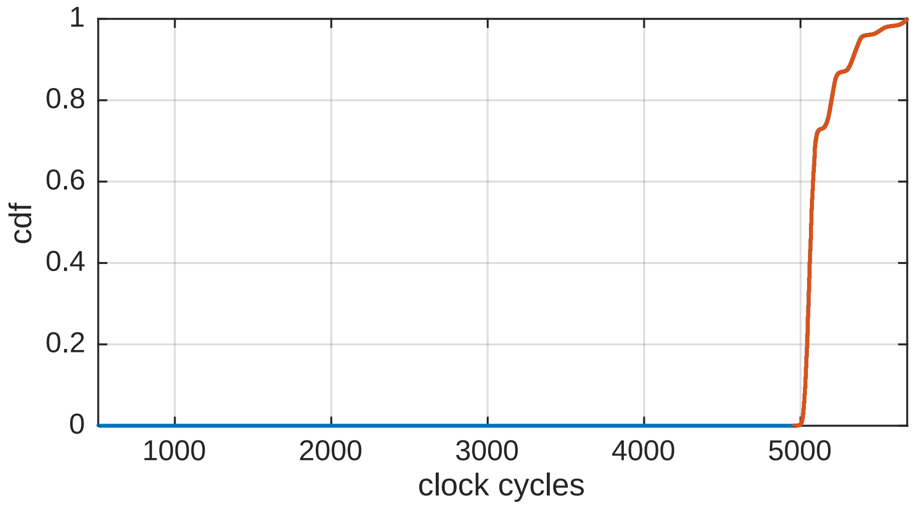


**Fig. 9.** Timing distribution cdfs. Early-abort sampler, intra-stage variability

## 6 Conclusion and Discussion

In this paper, we present a constant-time version of the Knuth-Yao sampling algorithm. We also present various optimizations to make the sampling algorithm many times faster than existing leakage resistant discrete Gaussian sampling algorithms. These optimizations do not require any special hardware and can be implemented on most modern processors.

We are aware that though this method does not require *large data memory* to store large precomputed tables, it requires a *larger program memory* than other methods, which is not so much problem for desktop computers as it is for devices with very limited resources. Future research will try to reduce the program memory by possibly tweaking the minimization procedure of Boolean functions or devising encoding schemes to reduce the storage of program memory. These methods may sacrifice its efficiency to some extent but will be suitable for devices with limited resources.

There are some simple optimizations that could be applied to make the method more efficient. For example, as our method does not require frequent external-memory accesses and has a high degree of parallelism, it can be exploited to design fast constant-time discrete Gaussian sampling on multi-core processors. Also, minimizations of the Boolean functions $f^i$ in Section 3.1 has a direct impact on the efficiency of the sampling algorithm. There is a possibility to use different tools to get a better minimization of the Boolean functions, which will immediately translate into a faster sampling process. Also, we can see from our results in Table 2 that generating pseudo-random numbers using SHA-512 takes most of the time in the whole sampling operation. It will be interesting to test the performance of the sampler using different pseudo-random number generators. We leave this for further research.

## 7 Acknowledgements

This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported in part by the Flemish Government, by the Hercules Foundation AKUL/11/19, and by the European Commission through ICT programme under contract through the Horizon 2020 research and innovation programme under contract No H2020-ICT-2014-644371 WITDOM, H2020-ICT-2014-644209 HEAT, H2020-ICT-2014-645622 PQCRYPTO and Cathedral ERC Advanced Grant 695305.

## References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In Henri Gilbert, editor, *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*, pages 553–572. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical ibe. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 98–115, Berlin, Heidelberg, 2010. Springer-Verlag.

[ABB$^+$16] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, pages 44–60. Springer International Publishing, Cham, 2016.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. In *Proceedings of the 25th USENIX Security Symposium*, pages 327–343, 2016.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ICALP'11, pages 403–415, Berlin, Heidelberg, 2011. Springer-Verlag.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 99–108, New York, NY, USA, 1996. ACM.

[BCD$^+$16] Joppe Bos, Craig Costello, Leo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1006–1018, New York, NY, USA, 2016. ACM.

[BCG$^+$14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. 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*, pages 402–417, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[BCNS15] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570, May 2015.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, July 2014.

[Bih97] Eli Biham. A fast new des implementation in software. In Eli Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97 Haifa, Israel, January 20–22 1997 Proceedings*, pages 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[BLL$^+$15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security,Auckland, New Zealand, November 29 – December 3, 2015, Proceedings, Part I*, pages 3–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[BLN$^+$16] Paulo S. L. M. Barreto, Patrick Longa, Michael Naehrig, Jefferson E. Ricardini, and Gustavo Zanon. Sharper ring-lwe signatures. Cryptology ePrint Archive, Report 2016/1026, 2016. `http://eprint.iacr.org/2016/1026`.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 868–886. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *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*, pages 523–552. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[CKN00] Jean-Sébastien Coron, Paul C. Kocher, and David Naccache. Statistics and secret leakage. In *Financial Cryptography*, volume 1962 of *LNCS*, pages 157–173. Springer, 2000.

[CNK04] Jean-Sébastien Coron, David Naccache, and Paul C. Kocher. Statistics and secret leakage. *ACM Trans. Embedded Comput. Syst.*, 3(3):492–508, 2004.

[DB15] Chaohui Du and Guoqiang Bai. Towards efficient discrete gaussian sampling for lattice-based cryptography. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, Sept 2015.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 40–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, 2014.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.

[GBHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 323–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, CA, USA, Stanford, CA, USA, 2009. AAI3382729.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, New York, NY, USA, 2008. ACM.

[HKR$^+$16] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, PP(99):1–1, 2016.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KS09] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009: 11th International Workshop Lausanne, Switzerland, September 6-9, 2009 Proceedings*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011: The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, pages 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *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*, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 738–755. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computational complexity*, 16(4):365–411, 2007.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew

Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 353–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 80–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Pes16] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, pages 153–170. Springer International Publishing, Cham, 2016.

[PZ03] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *eprint arXiv:quant-ph/0301141*, January 2003.

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *2017 Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, page 14, 2017.

[Reg04] Oded Regev. *New Lattice-based Cryptographic Constructions*, volume 51, pages 899–942. ACM, New York, NY, USA, November 2004.

[RRVV14] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Compact and side channel resistant discrete gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014. `https://eprint.iacr.org/2014/591.pdf`.

[RSD06] Chester Rebeiro, David Selvakumar, and A. S. L. Devi. Bitslice implementation of aes. In *Proceedings of the 5th International Conference on Cryptology and Network Security*, CANS'06, pages 203–212, Berlin, Heidelberg, 2006. Springer-Verlag.

[RVV14] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. High precision discrete gaussian sampling on fpgas. In *Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282*, pages 383–401, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[Saa15] Markku-Juhani O. Saarinen. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive, Report 2015/953, 2015. `http://eprint.iacr.org/2015/953`.

[Sho97] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, 26:1484, 1997.

[WHCB13] Patrick Weiden, Andreas Hülsing, Daniel Cabarcas, and Johannes Buchmann. Instantiating treeless signature schemes. Cryptology ePrint Archive, Report 2013/065, 2013. `https://eprint.iacr.org/2013/065.pdf`.