FACCT: FAst, Compact, and Constant-Time Discrete Gaussian Sampler over Integers

Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad

Abstract—The discrete Gaussian sampler is one of the fundamental tools in implementing lattice-based cryptosystems. However, a naive discrete Gaussian sampling implementation suffers from side-channel vulnerabilities, and the existing countermeasures usually introduce significant overhead in either the running speed or the memory consumption.
In this paper, we propose a fast, compact, and constant-time implementation of the binary sampling algorithm, originally introduced in the BLISS signature scheme. Our implementation adapts the Rényi divergence and the transcendental function polynomial approximation techniques. The efficiency of our scheme is independent of the standard deviation, and we show evidence that our implementations are either faster or more compact than several existing constant-time samplers. In addition, we show the performance of our implementation techniques applied to and integrated with two existing signature schemes: qTesla and Falcon. On the other hand, the convolution theorems are typically adapted to sample from larger standard deviations, by combining samples with much smaller standard deviations. As an additional contribution, we show better parameters for the convolution theorems.

Index Terms—Lattice-based crypto, discrete Gaussian sampling, constant-time, implementation, efficiency

1 Introduction

Various lattice-based cryptosystems require a discrete Gaussian sampler over integers (particularly centered at 0) as the subroutine during the implementation, including lattice-based signature schemes [1], [2] and public-key encryptions [3], [4], [5]. However, typical discrete Gaussian sampling implementations usually suffer from side-channel vulnerabilities. The straightforward Knuth-Yao [6] or binary-search Cumulative Distribution Table (CDT) [7] approaches are not constant-time, due to the branch conditions. For the binary sampling algorithm from the BLISS signature [1], the non-constant time implementation recently became the target of several side-channel attacks [8], [9], [10], [11] to recover the signing key. Therefore, it is important to implement discrete Gaussian samplers in cryptosystems, using constant-time algorithms.

Unfortunately, existing constant-time sampling methods are usually inefficient in either the running speed or the memory consumption. Let σ be the standard deviation of the discrete Gaussian distribution. For typical countermeasures applied to table-based sampling algorithms, such as the full-table access CDT approach [12], the running time is proportional to the table size $O(\tau\sigma)$, where τ is the tail-cut factor (typically about 10–12). Thus, the straightforward constant-time sampling algorithms are inefficient in both timing and memory consumption for large σ. To handle this scenario more efficiently, typically one employs a sampler (called a *base sampler*) to generate samples from a much smaller standard deviation σ_0, then use an *expander* to combine those samples together into a sample with the larger standard deviation σ. However, two major approaches for expander algorithms, namely the binary sampling implementation with constant-time countermeasures [9], [10], [13] and the convolution scheme [14], [15], both have drawbacks: For the first binary sampling expander approach, existing constant-time countermeasures either introduce significant overhead in the running speed [9], [10] or add large look-up tables [13]. The convolution expander approach can achieve σ_0 about $O(\log\log\sigma)$ in theory, but the total running time is still relatively large in practice, due to the fact that the base sampler needs to run 2^l times to generate a sample with standard deviation σ, where l is the number of convolution levels. For example, in [14], [16], where $\sigma_0 \approx 6.18$ and $l = 2$, the table-based base sampler has about 60 table entries, and to generate each sample with standard deviation σ, the sampling scheme needs to fully access this table 4 times. A recent bitslicing base sampler implementation [17] significantly improve the running speed, but at the expense of huge code size. Even worse, for cryptosystems requiring samples from several different standard deviations, such as [18], the two existing methods need to implement different tables or base samplers for each σ.

Due to the difficulty of implementing the sampler both efficiently and securely, some recent signature schemes [19] moved away from using the discrete Gaussian distribution, at the expense of larger signature size.

1.1 Contribution

In this paper, we introduce several new constant-time implementation techniques to address the above efficiency issues in existing discrete Gaussian sampling implementations. In particular, we make the following contributions:

- Our main contribution is to show that instead of storing many pre-computed $\exp(x)$ evaluations [13] or combining many Bernoulli samples [9], [10], the $\exp(x)$ polynomial approximation techniques with a carefully chosen precision can achieve faster and

- *Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad are with the Faculty of Information Technology, Monash University, Clayton VIC 3800, Australia.*
E-mail: {raymond.zhao,ron.steinfeld,amin.sakzad}@monash.edu

more compact constant-time implementations of the binary sampling expander. To minimise the required polynomial approximation precision, we show how to apply the Rényi divergence analysis to the binary sampling algorithm. Previous works on the Rényi divergence used a different order [20], only applied this technique to the rejection in the BLISS signing algorithm [21], or applied to a different sampling method [22]. As opposed to [23], where the authors discussed the simple polynomial approximation to the $\exp(x)$ function but discarded it as inefficient in discrete Gaussian sampling, we show that with carefully chosen polynomial approximation parameters, our constant-time implementation techniques can actually be more efficient than other methods.
- We show that our scheme enjoys the property that the implementation efficiency is independent of the standard deviation. In addition, we show that our implementation techniques are flexible to integrate with existing cryptosystems, such as qTesla [2] and Falcon [24].
- As an additional independent contribution, we show how to adapt the Rényi divergence analysis to the convolution sampling algorithm and achieve smaller σ_0 for the base sampler, compared to the existing Kullback-Leibler Divergence (KLD) based algorithms [14], [16].

1.2 Comparison with Subsequent Works

In the NIST PQC Round 2 submission, qTesla implemented a new variant of the CDT sampler [2]. To distinguish between two different sampling algorithms of qTesla submitted to the NIST PQC Round 1 and Round 2, we use the notations R1 (Round 1) and R2 (Round 2) in following discussions, respectively. The qTesla-R2 sampling algorithm generates a batch of randomness and sorts them with the CDT table entries in constant-time, and the position of randomness in the sorted table will imply the result of samples. While this algorithm is efficient and consumes less randomness for small $\sigma \approx 10.2$–22.93 compared to the previous qTesla-R1 implementation [13], however, we show that this algorithm is not scalable in Section 5.1 since the performance becomes significantly worse for larger $\sigma \approx 215$. On the other hand, our FACCT sampler maintains good performance even for smaller σ, as we show in Section 5.2 that the qTesla-R2 Keygen can be accelerated by about 2.3x–2.8x by plugging in our sampling scheme.

A recent work [25] discussed the relative errors in discrete Gaussian samplers due to the floating-point arithmetic. We acknowledge the issues in [25] and provide a comprehensive relative error analysis of our FACCT sampler in Section 4.2.1. In addition, very recently a fully constant-time implementation of the BLISS signature scheme [26] adapted a variant of our FACCT sampling algorithm without floating-point arithmetics. The polynomial approximation in this implementation has the same degree 9 as our FACCT sampler and employs only the integer arithmetic, which is more suitable for architectures where constant-time hardware floating-point instructions (addition and multiplication) may not be available. However, the techniques in [26] require a different polynomial approximation for each different σ and therefore lose the σ-independent feature of our FACCT sampling algorithm.

2 Preliminaries

Let $\rho_\sigma(x) = \exp\left(-x^2/2\sigma^2\right)$ be the (continuous) Gaussian function with zero mean and standard deviation σ. We denote the corresponding discrete Gaussian distribution on integer lattices with center zero and standard deviation σ as: $\mathcal{D}_{\mathbb{Z},\sigma}(x) = \rho_\sigma(x) / \sum_{k\in\mathbb{Z}} \rho_\sigma(k)$. We omit the lattice notation (i.e. $\mathcal{D}_\sigma$) if sampling from $\mathbb{Z}$. We denote $\mathcal{D}_\sigma^+$ as the distribution of $x \hookleftarrow \mathcal{D}_\sigma$ for all $x \in \mathbb{Z}^+$ (i.e. $\mathcal{D}_\sigma^+(x) = \rho_\sigma(x) / \sum_{k\in\mathbb{Z}^+} \rho_\sigma(k)$). In addition, we denote the uniform distribution on set S as $\mathcal{U}(S)$ and the Bernoulli distribution with bias p as $\mathcal{B}_p$ (i.e. the probability distribution with $\Pr(X=1)=p$ and $\Pr(X=0)=1-p$). A distribution is B-bounded for some $B \in \mathbb{R}^+$, if its support is in the interval $[-B, B]$ [20]. Also, for a lattice Λ and any $\epsilon \in \mathbb{R}^+$, we denote the smoothing parameter $\eta_\epsilon(\Lambda)$ as the smallest $s \in \mathbb{R}^+$ such that $\rho_{1/(s\cdot\sqrt{2\pi})}\left(\Lambda^* \setminus \left\{\vec{0}\right\}\right) \leq \epsilon$, where Λ^* is the dual lattice of Λ: $\Lambda^* = \{\vec{w} \in \mathbb{R}^n : \forall \vec{x} \in \Lambda, \vec{x}\cdot\vec{w} \in \mathbb{Z}\}$ [27]. An upper bound on $\eta_\epsilon(\mathbb{Z})$ is given by [27]: $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2+2/\epsilon)/\pi}$.

Definition 1 (Relative Error). For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, the relative error between $\mathcal{P}$ and $\mathcal{Q}$ is defined as:

$$\Delta(\mathcal{P}||\mathcal{Q}) = \max_{x\in\mathrm{Supp}(\mathcal{P})} \frac{|\mathcal{P}(x) - \mathcal{Q}(x)|}{\mathcal{Q}(x)}.$$

Definition 2 (Kullback-Leibler Divergence [14]). For two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) \subseteq \mathrm{Supp}(\mathcal{Q})$, the Kullback-Leibler divergence (KLD) is defined as:

$$KL(\mathcal{P}||\mathcal{Q}) = \sum_{x\in\mathrm{Supp}(\mathcal{P})} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Definition 3 (Rényi Divergence [20], [21]). For two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) \subseteq \mathrm{Supp}(\mathcal{Q})$, the Rényi divergence (RD) of order $\alpha \in (1, +\infty)$ is defined as:

$$R_\alpha(\mathcal{P}||\mathcal{Q}) = \left(\sum_{x\in\mathrm{Supp}(\mathcal{P})} \frac{\mathcal{P}(x)^\alpha}{\mathcal{Q}(x)^{\alpha-1}}\right)^{\frac{1}{\alpha-1}}.$$

In addition, for $\alpha = +\infty$, we have:

$$R_\infty(\mathcal{P}||\mathcal{Q}) = \max_{x\in\mathrm{Supp}(P)} \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Definition 4 (Max-log Distance [15]). For two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, the max-log distance is defined as:

$$ML(\mathcal{P}||\mathcal{Q}) = \max_{x\in\mathrm{Supp}(\mathcal{P})} |\ln\mathcal{P}(x) - \ln\mathcal{Q}(x)|.$$

For tighter bounds, we use the following theorems in this paper:

Theorem 1 (Tail-cut Bound, Adapted from [20], Thm. 2.11). Let $\mathcal{D}'_\sigma$ be the B-bounded distribution of $\mathcal{D}_\sigma$ by cutting its tail. For M independent samples, we have $R_\infty\left(\left(\mathcal{D}'_\sigma\right)^M || \left(\mathcal{D}_\sigma\right)^M\right) \leq \exp(1)$ if $B \geq \sigma\cdot\sqrt{2\ln(2M)}$.

Theorem 2 (Relative Error Bound, Adapted from [21], Lemma 3 and Eq. 4). For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, we have:

$$R_\alpha(\mathcal{P}||\mathcal{Q}) \leq \left(1 + \frac{\alpha(\alpha-1)\cdot(\Delta(\mathcal{P}||\mathcal{Q}))^2}{2(1-\Delta(\mathcal{P}||\mathcal{Q}))^{\alpha+1}}\right)^{\frac{1}{\alpha-1}}.$$

The right-hand side is asymptotically equivalent to $1 + \alpha\cdot(\Delta(\mathcal{P}||\mathcal{Q}))^2/2$ as $\Delta(\mathcal{P}||\mathcal{Q}) \to 0$. In addition, if a signature scheme using M independent samples from $\mathcal{Q}$ is $(\lambda+1)$-bit secure, then the signature scheme sampling from $\mathcal{P}$ will be λ-bit secure if $R_{2\lambda}(\mathcal{P}||\mathcal{Q}) \leq 1 + 1/(4M)$.

Typically we have $M = m\cdot q_s$, where m is the dimension of the lattice and q_s is the number of queries.

3 Review of Discrete Gaussian Sampling Schemes

To sample from $\mathcal{D}_\sigma$ for large σ, typically one generates samples from a base sampler with much smaller standard deviations, then combines the samples together with an expander. We review two commonly used expanding approaches in this section: the binary sampling algorithm [1] and the convolution methods [14], [15].

3.1 Binary Sampling Method

The original binary sampling method was proposed by [1] in the BLISS signature scheme. Let $\sigma = k\sigma_0$, $k \in \mathbb{Z}^+$, and $\sigma_0 = \sqrt{1/(2\ln 2)}$. This algorithm samples from $\mathcal{D}_\sigma^+$ by first generating a sample $x \hookleftarrow \mathcal{D}_{\sigma_0}^+$ from the base sampler and an integer $y \hookleftarrow \mathcal{U}(\{0,\ldots,k-1\})$, then performing a rejection sampling on $z = kx + y$, with the acceptance rate:

$$p = \exp\left(\frac{-y(y+2kx)}{2\sigma^2}\right). \tag{1}$$

To generate negative samples, one can sample and apply a random sign bit, with the expection of rejection with probability $1/2$ when $z = 0$.

Theorem 3 (Adapted from [1], Thm. 6.6). Given $x \hookleftarrow \mathcal{D}_{\sigma_0}^+$ and $y \hookleftarrow \mathcal{U}(\{0,\ldots,k-1\})$, the probability to output some integer $z = kx + y$ is proportional to:

$$\begin{aligned}\rho_{\sigma_0}(x)\cdot p &= \exp\left(-\frac{x^2}{2\sigma_0^2} - \frac{-y(y+2kx)}{2(k\sigma_0)^2}\right)\\ &= \exp\left(-\frac{(kx+y)^2}{2(k\sigma_0)^2}\right)\\ &= \rho_{k\sigma_0}(z)\\ &= \rho_\sigma(x).\end{aligned}$$

The rejection framework of the binary sampling algorithm is shown in Fig.1. The rejection sampling itself will not leak any secret information, if the underlying base sampler and the Bernoulli sampler are side-channel resistant. Unfortunately, to achieve efficient algorithms, the original sampler implementations in the BLISS signature are not constant-time (see Fig.2 and Fig.3, respectively). When attacking signature schemes similar to the BLISS, the attacker can gather the discrete Gaussian vectors, or the intermediate base samples and Bernoulli samples, by exploiting the side-channels, such as the cache [8], or timing and power [10],

Output: A sample from $\mathcal{D}_\sigma^+$.
function BINARYSAMPLER(k)
  Let $x \hookleftarrow \mathcal{D}_{\sigma_0}^+$.
  Let $y \hookleftarrow \mathcal{U}(\{0,\ldots,k-1\})$.
  Let $z = kx + y$.
  Let $t = y(y+2kx)$.
  Let $b \hookleftarrow \mathcal{B}_{\exp(-t/2\sigma^2)}$.
  if $b = 0$ **then**
    Restart BinarySampler.
  end if
  return z.
end function

Fig. 1. Binary sampling scheme [1].

Output: A sample from $\mathcal{D}_{\sigma_0}^+$.
function BASESAMPLER
  Sample $b \hookleftarrow \mathcal{U}(\{0,1\})$.
  if $b = 0$ **then**
    return 0.
  end if
  $i = 1$
  while true do
    Sample $(b_1, b_2, \ldots, b_{2i-1}) \hookleftarrow (\mathcal{U}(\{0,1\}))^{2i-1}$.
    if $(b_1, b_2, \ldots, b_{2i-2}) \neq (0,0,\ldots,0)$ **then**
      Restart BaseSampler.
    end if
    if $b_{2i-1} = 0$ **then**
      return i.
    end if
    $i = i + 1$.
  end while
end function

Fig. 2. Base sampler from BLISS [1].

Input: Integer $t = y(y+2kx)$ with $0 \leq t < 2^l$ and binary form $t = t_{l-1}\ldots t_0$, where $x \hookleftarrow \mathcal{D}_{\sigma_0}^+$ and $y \hookleftarrow \mathcal{U}(\{0,\ldots,k-1\})$. Pre-computed table $p_i = \exp(-2^i/2\sigma^2)$ for $i < l$.
Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2)$.
function BERNOULLISAMPLER(t)
  for $i = l-1$ **downto** 0 **do**
    if $t_i = 1$ **then**
      Sample $a \hookleftarrow \mathcal{B}_{p_i}$.
      if $a = 0$ **then**
        return 0.
      end if
    end if
  end for
  return 1.
end function

Fig. 3. Bernoulli sampler from BLISS [1].

and then recover the signing key by using the leaked information. These attacks only require about several thousand signatures and the corresponding samples to succeed.

To mitigate these side-channel attacks, several efforts have been proposed. We review them now.

Input: Integer $t = y(y+2kx)$ with $0 \leq t < 2^l$ and binary form $t = t_{l-1} \dots t_0$, where $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$ and $y \hookleftarrow \mathcal{U}(\{0, \dots, k-1\})$. Pre-computed table $p_i = \exp\left(-2^i/2\sigma^2\right)$ for $i < l$.
Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2)$.
function BERNOULLISAMPLER(t)
Let $r = 1$.
for $i = l-1$ **downto** 0 **do**
Sample $a \hookleftarrow \mathcal{B}_{p_i}$.
Set $r = r \cdot (1 - t_i + at_i)$.
end for
return r.
end function

Fig. 4. Constant-time Bernoulli sampler [9], [10].

3.2 Existing Timing/Cache Attack Countermeasures for the Binary Sampling Method

3.2.1 Random Shuffle

One commonly used heuristic countermeasure is performing the Fisher-Yates random shuffle (or Knuth shuffle) [6], to mask the relation between the retrieved side-channel information of the samples and the secret, after performing non-constant time sampling schemes [28], [29]. However, in the above mentioned attacking scenarios, the random permutation cannot totally hide the statistical features of the distributions in the attacked vector. By performing statistical analysis, it was shown in [30] that an attacker only requires marginally larger yet still practical number of samples to rearrange the coordinates and "undo" the shuffle.

3.2.2 Constant-time Base/Bernoulli Sampler

The base sampler can be implemented in constant-time, by using a full-table access Cumulative Distribution Table (CDT) sampler [12]. A recent work [31] suggested using a binary search CDT sampler with constant number of iterations $O(\log B)$ on hardware, where B is the tail-cut bound. However, the memory access in this approach is not constant, which might cause potential cache timing leakage in software implementations [17]. On the other hand, for the table-based Bernoulli sampler, several works [8], [9], [10] suggested the countermeasure of removing the branches and performing full-table access (see Fig.4). However, this countermeasure adds significant overhead, since it requires additional randomness for each table entry. A recent lattice-based signature scheme in the NIST PQC submission [32], qTesla-R1 [13], suggested a more efficient approach that the sampler computes the bias p in (1) by multiplying table entries from each subtable based on the binary representation of the input, where every subtable $\mathcal{B}_i$ has $32 \cdot 8 = 256$ bytes (see Fig.5). However, although the number of iterations in this sampler is constant, the memory access pattern depends on the size of the underlying CPU cachelines. This could cause a potential leakage via cache timing side-channels on some architectures.

3.3 Convolution Methods

Previous works [14], [16] suggested applying the following KLD-based convolution theorem to construct discrete Gaussian sampling algorithms:

Input: Integer $t = y(y+2kx)$ with $0 \leq t < 2^{15}$, where $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$ and $y \hookleftarrow \mathcal{U}(\{0, \dots, k-1\})$. Pre-computed Bernoulli table entries $\mathcal{B}_{i,j} = \exp\left(-2^{(j \cdot 32^i)}/2\sigma^2\right)$, where $i, j \in \mathbb{Z}^+$, $0 \leq i < 3$, and $0 \leq j < 32$. Each $\mathcal{B}_{i,j}$ has 8 bytes.
Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2)$.
function BERNOULLISAMPLER(t)
Sample $r \hookleftarrow \mathcal{U}\left(\{0,1\}^{62}\right)$.
Let $c = 2^{62}$.
Let $s = t$.
for $i = 0$ **to** 2 **do**
Set $c = c \cdot \mathcal{B}_{i, s \bmod 32}$.
Set $s = s/32$.
end for
if $r \geq \lfloor c \rceil$ **then**
return 0.
else
return 1.
end if
end function

Fig. 5. Bernoulli sampler with constant number of iterations [13].

Output: A sample from $\mathcal{D}_\sigma$, where $\sigma \approx 215$.
function CONVOLUTIONSAMPLER
Sample $x_1, x_2, x_3, x_4 \hookleftarrow \mathcal{D}_{\sigma_0}$, where $\sigma_0 \approx 6.18$.
Let $y = (x_1 + 3x_2) + 11 \cdot (x_3 + 3x_4)$.
return y.
end function

Fig. 6. KLD-based convolution sampling scheme [14], [16].

Theorem 4 (KLD-based Convolution Theorem, Adapted from [14], Lemma 3). Let $x_1 \hookleftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}$ and $x_2 \hookleftarrow \mathcal{D}_{k\mathbb{Z},\sigma_2}$ for some $\sigma_1, \sigma_2 \in \mathbb{R}^+$. Let $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$. For any $\epsilon \in (0, 1/2)$, if $\sigma_1 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$ and $\sigma_3 \geq \eta_\epsilon(k\mathbb{Z})/\sqrt{2\pi}$, then the distribution $\mathcal{P}$ of $x_1 + x_2$ satisfies:

$$KL(\mathcal{P}||\mathcal{D}_\sigma) \leq 2\left(1 - \left(\frac{1+\epsilon}{1-\epsilon}\right)^2\right)^2 \approx 32\epsilon^2.$$

For the deviation $\sigma \approx 215$ in the BLISS-I parameter set, one can generate $x_1, x_2 \hookleftarrow \mathcal{D}_{\sigma_1}$ and compute $x_1 + k_1 x_2$, where $\sigma_1 = \sigma/\sqrt{1+k_1^2} \approx 19.53$ and $k_1 = 11$. Sampling from $\mathcal{D}_{\sigma_1}$ can be further decomposed into $x_3 + k_2 x_4$, where $x_3, x_4 \hookleftarrow \mathcal{D}_{\sigma_2}$, $\sigma_2 = \sigma_1/\sqrt{1+k_2^2} \approx 6.18$, and $k_2 = 3$ (see Fig.6). If the sampling algorithm of $\mathcal{D}_{\sigma_2}$ (or $\mathcal{D}_{\sigma_1}$) is constant-time, then the whole sampling scheme will be constant-time. To sample from $\mathcal{D}_{\sigma_2}$, recent works [17], [33] adapted the bitslicing method to implement the Knuth-Yao algorithm [34] more efficiently in constant-time, compared to the previous full-table access CDT approach.

Meanwhile, a recent work [15] proposed the following max-log based convolution theorems:

Theorem 5 (Adapted from [15], Cor. 4.1). Let $\vec{z} = (z_1, \dots, z_n) \in \mathbb{Z}^n$ be a nonzero vector with $\gcd(z_1, \dots, z_n) = 1$ and $\vec{\sigma} = (\sigma_1, \dots, \sigma_n) \in \mathbb{R}^n$ with $\sigma_i \geq \|\vec{z}\|_\infty \cdot \eta_\epsilon(\mathbb{Z})/\sqrt{\pi}$ for all $i \leq n$. Let $\vec{y} \hookleftarrow \left(\mathcal{D}'_{\sigma_i}\right)^n$, with $ML\left(\mathcal{D}'_{\sigma_i}||\mathcal{D}_{\sigma_i}\right) \leq \mu_i$ for all i. Let $\sigma^2 = \sum z_i^2 \sigma_i^2$

and $\mathcal{P}$ be the distribution of $\sum z_i y_i$. Then $ML(\mathcal{P}||\mathcal{D}_\sigma) \leq 2\epsilon + \sum \mu_i$.

Theorem 6 (Adapted from [15], Cor. 4.2). Let $x_1 \hookleftarrow \mathcal{D}'_{\mathbb{Z},\sigma_1}$ and $x_2 \hookleftarrow \mathcal{D}'_{k\mathbb{Z},\sigma_2}$ for some $\sigma_1, \sigma_2 \in \mathbb{R}^+$. Let $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$. If $\sigma_1 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$, $\sigma_3 \geq \eta_\epsilon(k\mathbb{Z})/\sqrt{2\pi}$, $ML\left(\mathcal{D}'_{\mathbb{Z},\sigma_1}||\mathcal{D}_{\mathbb{Z},\sigma_1}\right) \leq \mu_1$, and $ML\left(\mathcal{D}'_{k\mathbb{Z},\sigma_2}||\mathcal{D}_{k\mathbb{Z},\sigma_2}\right) \leq \mu_2$, then the distribution $\mathcal{P}$ of $x_1 + x_2$ satisfies $ML(\mathcal{P}||\mathcal{D}_\sigma) \leq 4\epsilon + \mu_1 + \mu_2$.

4 Proposed Constant-time Implementation Techniques

4.1 Directly Approximating the Exp Function

The Bernoulli bias p in (1) can be directly computed within double precision (53 bits), if the RD-based relative error bound (Theorem 2) is adapted [21]. Falcon [24], a recent lattice-based signature scheme in the NIST PQC submission, applied this approach to compute the rejection bias when sampling from the arbitrary-centered discrete Gaussian distribution, by using a rational function approximation of $\exp(x)$, similar to the implementation in the C standard library (see Fig.7). However, the floating-point division instructions on the Intel CPUs have various latency and throughput [35]. Furthermore, the compiler may replace the division operation with its own arithmetic library routine, which may not be constant-time [36]. Therefore, the division arithmetic should be generally avoided in constant-time implementation.

Another classical method to compute the $\exp(x)$ is the Padé approximation [21], which uses $P(x)/Q(x)$ to approximate $\exp(x)$ for some polynomials $P(x) = Q(-x)$. For the BLISS signature scheme, to satisfy the relative error bound by Theorem 2, $P(x)$ and $Q(x)$ need to be at least degree 7 by our experiments in the `sagemath` tool. To compare $u < P(x)/Q(x)$ for some $u \hookleftarrow \mathcal{U}([0,1))$ in the rejection step of the binary sampling scheme, one may instead perform the comparison of $u \cdot Q(x) < P(x)$ to avoid the floating-point division. However, it is unclear how to choose the precisions of u and the $u \cdot Q(x)$ multiplication operation for this method. Another issue is how to efficiently implement this approach in constant-time, since the implementation may involve either a high precision floating-point multiplication for $u \cdot Q(x)$ or computing the multiplication between the mantissas of u and $Q(x)$ with integer arithmetics larger than 64 bits.

We compute the $\exp(x)$ by evaluating a polynomial at point x instead, where only the floating-point additions and multiplications are involved. Both the addition and the multiplication instructions on the Intel CPUs have constant latency and throughput [35]. To find such an $\exp(x)$ approximation with sufficient precision, we use the following approach:

1) Let $t = y(y + 2kx)$. First, we observe that since $\sigma_0 = \sqrt{1/(2\ln 2)}$ and $\sigma = k\sigma_0$, the Bernoulli bias p in (1) can be re-written as:

$$p = \exp\left(-t/2\sigma^2\right) = \exp\left(-\ln 2 \cdot t/k^2\right) = 2^{-t/k^2}.$$

Therefore, we can find a polynomial approximation of $2^{-t/k^2}$ for $t \geq 0$.

Input: $x \in \mathbb{R}$, such that $|x| \leq \ln 2$.
Output: e^x with about 50-bit precision.
function EXP(x)
Let $p_1 = 1.66666666666666019037 \cdot 10^{-1}$.
Let $p_2 = -2.77777777770155933842 \cdot 10^{-3}$.
Let $p_3 = 6.61375632143793436117 \cdot 10^{-5}$.
Let $p_4 = -1.65339022054652515390 \cdot 10^{-6}$.
Let $p_5 = 4.13813679705723846039 \cdot 10^{-8}$.
Let $s = x/2$.
Let $t = s^2$.
Let $c = s - t \cdot (p_1 + t \cdot (p_2 + t \cdot (p_3 + t \cdot (p_4 + t \cdot p_5))))$.
Let $r = 1 - ((s \cdot c)/(c - 2) - s)$.
return r^2.
end function

Fig. 7. Rational function approximation algorithm of $\exp(x)$ [24].

2) Second, we adapt the method from [37]. Let $a = -t/k^2$. We get:

$$2^a = 2^{\lfloor a \rfloor + z} = 2^{\lfloor a \rfloor} \cdot 2^z,$$

for $0 \leq z < 1$, where z is the remaining part of rounding operation. We can directly get the multiplication with $2^{\lfloor a \rfloor}$ by changing the exponent of a floating-point variable. To approximate 2^z, we use the `sollya` tool [38] to find a polynomial with sufficient number of terms, such that the minimax error is within the RD-based relative error bound.

According to the manual of the `sollya` tool [39], we use the following three functions to get such a polynomial and verify its precision:

- The $\text{guessdegree}(f, I, \delta, \omega)$ function finds the minimal degree sufficient for the polynomial approximation P of function f over the interval I, such that $\|P\omega - f\|_\infty < \delta$. For example, we use the command `guessdegree(1, [0,1], 1b-45, 1/2^x)` to estimates the minimal degree of polynomial approximation $P(x)$ over the interval $[0, 1]$, such that:

$$\|P/2^x - 1\|_\infty < 2^{-45} \implies \Delta(P||2^x) < 2^{-45}.$$

- The $\text{fpminimax}(f, n, L, I, \text{floating}, \text{relative})$ function performs the heuristic from [40] to find a degree-n polynomial approximation P of function f over the interval I, such that P has the minimal minimax relative error, with the i-th floating-point coefficient c_i having precision L_i for all $i \leq n$. For example, we use the command `fpminimax(2^x, 9, [|1,D...|], [0,1], floating, relative)` to find the polynomial approximation $P(x)$ of 2^x over the interval $[0, 1]$, with degree 9 (the result from the previous guessdegree command) and double precision coefficients ("D" represents double precision in this command). To make sure $P(0) = 1$, we set $L_0 = 1$ (1-bit precision), which results in coefficient $c_0 = 1$.
- The $\text{supnorm}(p, f, I, \text{relative}, accuracy)$ function computes the interval bound $r = [l, u]$ for the supremum norm of the relative error $\Delta = |p/f - 1|$

over the interval I, such that $\sup_{x\in I}\{\Delta(x)\} \subseteq r$ and $0 \leq |u/l - 1| \leq accuracy$. For example, we use the command `supnorm(P, 2^x, [0,1], relative, 1b-128)` to verify $\Delta(P||2^x)$ over the interval $[0,1]$ is smaller than the required relative error bound, where P is the polynomial approximation computed in the previous fpminimax command.

4.2 FACCT Algorithm

Our constant-time Bernoulli sampler adapting the $\exp(x)$ approximation approach above is shown in Fig.8. Let the standard deviation $\sigma = k\sigma_0$, where $k \in \mathbb{Z}^+$ and $\sigma_0 = \sqrt{1/(2\ln 2)}$. Let $P(z)$ be the polynomial approximation of 2^z with δ_P-bit precision for $0 \leq z < 1$. Given an integer $t = y\,(y + 2kx)$, where $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$ with tail-cut bound B and $y \hookleftarrow \mathcal{U}\,(\{0,\ldots,k-1\})$, this algorithm generates a sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2) = 2^{-t/k^2}$. We assume an IEEE-754 floating-point value $f \in (0,1]$ with (δ_f+1)-bit precision is represented by $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$, where integer $mantissa$ has δ_f bits and $exponent \in \mathbb{Z}^-$.

4.2.1 FACCT Relative Error Analysis

Here, we analyse the relative error of Fig.8. Since the algorithm will output 1 when $f = 1.0$, we only consider the case when $f \in (0,1)$, which implies $exponent < 0$. Let $\mathcal{P}_{\text{FACCT}}$ and $\mathcal{P}_{\text{IDEAL}}$ represent the distribution of the FACCT Bernoulli sampler and the ideal Bernoulli sampler, respectively. Since $a = -t/k^2$ and $z = a - \lfloor a \rfloor$, we have:

$$\mathcal{P}_{\text{IDEAL}}(\lfloor a \rfloor, z) = \exp(-t/2\sigma^2) = 2^a = 2^{z+\lfloor a \rfloor}.$$

Theorem 7 (Adapted from [37], Def. 5.1 and Eq. 5.7). The absolute error between an accurate polynomial evaluation $P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ and the evaluation $H(x)$ by using Horner's rule with δ-bit precision floating-point arithmetic is:

$$|H(x) - P(x)| \leq \gamma_{2n} \cdot \sum_{i=0}^{n} |a_i| \cdot |x|^i,$$

where $\gamma_{2n} \approx 2n \cdot 2^{-\delta}$ if $2n \ll 1/2^{-\delta}$.

The polynomial approximation P from the `sollya` tool is evaluated by using Horner's rule (see Fig.10 for an example). Assume a degree-n polynomial approximation P only has positive coefficients. Since we use (δ_f+1)-bit precision floating-point arithmetic to compute $P(z)$ for $0 \leq z < 1$, by adapting Theorem 7, we have:

$$\begin{aligned}
\mathcal{P}_{\text{FACCT}}(\lfloor a \rfloor, z) &= \frac{\left(mantissa + 2^{\delta_f}\right)}{2^{\delta_f+1}} \cdot \frac{2^{l+exponent+1}}{2^l} \\
&= \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent} \\
&= f \\
&\leq \left(1 + 2n \cdot 2^{-(\delta_f+1)}\right)\left(P(z) \cdot 2^{\lfloor a \rfloor}\right),
\end{aligned}$$

where the last inequality follows because $|H(z)/P(z) - 1| \leq \gamma_{2n}$ and $\gamma_{2n} \approx 2n \cdot 2^{-(\delta_f+1)}$ due to Theorem 7 (The multiplication with $2^{\lfloor a \rfloor}$ does not change the error since we directly change the exponent of a floating-point variable). Then, the relative error Δ between $\mathcal{P}_{\text{FACCT}}$ and $\mathcal{P}_{\text{IDEAL}}$ is:

$$\begin{aligned}
\Delta &= \max_{\lfloor a \rfloor, z} \left| \frac{\mathcal{P}_{\text{FACCT}}(\lfloor a \rfloor, z)}{\mathcal{P}_{\text{IDEAL}}(\lfloor a \rfloor, z)} - 1 \right| \\
&\leq \max_{\lfloor a \rfloor, z} \left| \frac{\left(1 + 2n \cdot 2^{-(\delta_f+1)}\right)\left(P(z) \cdot 2^{\lfloor a \rfloor}\right)}{2^{z+\lfloor a \rfloor}} - 1 \right| \\
&\leq \left(1 + n \cdot 2^{-\delta_f}\right)\left(1 + 2^{-\delta_P}\right) - 1 \quad \text{(by definition of } \delta_P) \\
&= 2^{-\delta_P} + n \cdot \left(2^{-\delta_f} + 2^{-(\delta_P+\delta_f)}\right). \qquad (2)
\end{aligned}$$

We also need to make sure that $l + exponent + 1 \geq 0$ during the comparison in Fig.8. Let Δ be the relative error in (2). Since $a = -t/k^2$, by definitions of $exponent$ and Δ from (2), we have:

$$\begin{aligned}
exponent &\geq \left\lfloor \log_2\left((1-\Delta) \cdot 2^{-t/k^2}\right) \right\rfloor \\
&\geq \lfloor -1 - t/k^2 \rfloor \quad \text{(we make } \Delta \leq 1/2) \\
&\geq \left\lfloor -1 - \frac{y(y+2kx)}{k^2} \right\rfloor \quad \text{(by definition of } t) \\
&\geq \left\lfloor -1 - \frac{y^2}{k^2} - \frac{2kxy}{k^2} \right\rfloor \\
&\geq -2B - 2. \quad \text{(by definitions of } x \text{ and } y)
\end{aligned}$$

Therefore, if $l + exponent + 1 \geq 0$, we have:

$$l \geq 2B + 1. \qquad (3)$$

To ensure that the compiler will not replace any floating-point arithmetic with its own library implementation, we manually write the arithmetic in the source code by using constant-time instructions with the Intel intrinsics. This also

Input: Deviation $\sigma = k\sigma_0$, where $k \in \mathbb{Z}^+$ and $\sigma_0 = \sqrt{1/(2\ln 2)}$. Integer $t = y\,(y + 2kx)$, where $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$ with tail-cut bound B and $y \hookleftarrow \mathcal{U}\,(\{0,\ldots,k-1\})$. Polynomial approximation $P(z)$ of 2^z with δ_P-bit precision for $0 \leq z < 1$. Bit length $l \geq 2B + 1$.

Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2) = 2^{-t/k^2}$.

function BERNOULLISAMPLER(t)
  Let $a = -t/k^2$.
  Let $z = a - \lfloor a \rfloor$.
  Evaluate $s = P(z)$ on point z.
  Let $f = s \cdot 2^{\lfloor a \rfloor}$.
  Let represent f by $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$, with δ_f-bit $mantissa$.
  Sample $r_m \hookleftarrow \mathcal{U}\left(\{0,1\}^{\delta_f+1}\right)$.
  Sample $r_e \hookleftarrow \mathcal{U}\left(\{0,1\}^{l}\right)$.
  if $\left(r_m < mantissa + 2^{\delta_f}\right.$ **and** $\left.r_e < 2^{l+exponent+1}\right)$ **or** $f = 1.0$ **then**
    return 1.
  else
    return 0.
  end if
end function

Fig. 8. FACCT Bernoulli sampler.

Input: Center $c'_F \in (0, 1/2]$.
Output: A sample from $\mathcal{D}_{c'_F,\sigma}$.
  function BINARYSAMPLEREX(c'_F, k)
    Let $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$.
    Let $y \hookleftarrow \mathcal{U}(\{0, \ldots, k-1\})$.
    Let $s \hookleftarrow \mathcal{U}(\{-1, 1\})$.
    Let $\delta = \lceil kx + sc'_F \rceil - kx - sc'_F$.
    if $y + \delta \geq k$ **then**
      Restart BinarySamplerEx.
    end if
    Let $z = \lceil kx + sc'_F \rceil + y$.
    Let $t = 2kx(y + \delta) + (y + \delta)^2$.
    Let $b \hookleftarrow \mathcal{B}_{\exp(-t/2\sigma^2)}$.
    if $b = 0$ **then**
      Restart BinarySamplerEx.
    end if
    return $s \cdot z$.
  end function

Fig. 9. Non-zero centered binary sampling scheme [41].

enables the Single Instruction Multiple Data (SIMD) instruction sets, such as the AVX2, which computes 4x double precision floating-point arithmetic in parallel.

Compared with the previous table-based constant-time Bernoulli sampling techniques [8], [9], [10], [13], where the number of table entries is proportional to the bit length of t, our implementation is more compact in terms of the memory consumption, since we only need to store a small number of polynomial coefficients. For example, in the BLISS-I parameter set, $k = 254$, which implies $0 \leq t < 2^{21}$. This requires at least 21 table entries in the previous techniques, compared to only 9 coefficients for about 45-bit precision in our implementation (see Section 4.5). Also, our implementation is more efficient for large standard deviations, since the code is independent of σ (assuming $-1/k^2$ is a pre-computed constant), while the number of iterations (proportional to the number of table entries) relies on k in previous table-based approaches. In addition, if the application requires samples from several different standard deviations, our implementation does not need additional pre-computed tables for each different k.

4.3 Non-zero Centered Discrete Gaussian Sampling

In this section, we discuss how to generalise our FACCT sampler to sample from non-zero centered discrete Gaussian distributions. We denote the non-zero centered discrete Gaussian distribution on integers as: $\mathcal{D}_{c,\sigma}(x) = \rho_\sigma(x - c) / \sum_{k \in \mathbb{Z}} \rho_\sigma(k - c)$, where the center $c \in \mathbb{R}$. A recent work [41] extended the binary sampling scheme to arbitrary non-zero centered discrete Gaussians as follows: For any center $c \in \mathbb{R}$, sampling from $\mathcal{D}_{c,\sigma}$ is equivalent to sampling from $\mathcal{D}_{c_F,\sigma} + \lfloor c \rfloor$, where $c_F = c - \lfloor c \rfloor \in [0, 1)$ is the fractional part of c. In addition, for $1/2 \leq c_F < 1$, sampling from $\mathcal{D}_{c_F,\sigma}$ is equivalent to sampling from $1 - \mathcal{D}_{c'_F,\sigma}$ where $c'_F = 1 - c_F \in (0, 1/2]$. A modified binary sampling scheme [41] can then be adapted to sample from $\mathcal{D}_{c'_F,\sigma}$ with any $c'_F \in (0, 1/2]$ (see Fig.9).

Since the Bernoulli bias in Fig.9 still has the form $\exp(-t/2\sigma^2)$ where $\sigma = k\sigma_0$, $k \in \mathbb{Z}^+$, and $\sigma_0 = \sqrt{1/(2\ln 2)}$, one can easily adapt our FACCT Bernoulli sampler in Fig.8 to implement the non-zero centered discrete Gaussian sampling scheme in [41]. However, the average number of trials in Fig.9 has the upper-bound $\left(\sigma^2 / (\sigma_0 \sigma - \sigma_0^2)\right) \cdot \left(\rho_{\sigma_0}(\mathbb{Z}^+) / \left(\sigma\sqrt{\pi/2} - 1\right)\right)$ [41] and therefore the rejection rate of sampling may reveal σ.

4.4 Concrete Rényi Divergence Based Convolution Sampling

Previous works [21] only implied the potentially tighter parameters for the convolution theorem based samplers by adapting the Rényi divergence. In this section, we discuss the concrete parameter choice for the RD-based convolution sampling scheme.

Theorem 8 (Adapted from [21], Lemma 4). For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, we have:

$$R_\alpha(\mathcal{P}||\mathcal{Q}) \leq \left(1 + \frac{\alpha(\alpha-1) \cdot \left(e^{ML(\mathcal{P}||\mathcal{Q})} - 1\right)^2}{2\left(2 - e^{ML(\mathcal{P}||\mathcal{Q})}\right)^{\alpha+1}}\right)^{\frac{1}{\alpha-1}}.$$

The right-hand side is asymptotically equivalent to $1 + \alpha \cdot (ML(\mathcal{P}||\mathcal{Q}))^2/2$ as $ML(\mathcal{P}||\mathcal{Q}) \to 0$.

Recall that $ML(\mathcal{P}||\mathcal{Q}) \approx \Delta(\mathcal{P}||\mathcal{Q})$ when $\Delta(\mathcal{P}||\mathcal{Q}) \to 0$ (Lemma 4.2, [15]). Also, in the convolution sampler adapting Theorem 5, typically $\vec{z} = (k-1, k)$ for some $k \geq 4$ [15]. Therefore, by applying Theorem 2, we provide the following concrete RD-based parameter choice lemmas:

Lemma 1. Let $x_1, x_2 \hookleftarrow \mathcal{D}'_{\sigma_0}$, with $\sigma_0 = \sigma/\sqrt{(k-1)^2 + k^2}$ for some $\sigma \in \mathbb{R}^+$ and $k \geq 4$. If $\sigma_0 \geq k\eta_\epsilon(\mathbb{Z})/\sqrt{\pi}$ and $\Delta(\mathcal{D}'_{\sigma_0}||\mathcal{D}_{\sigma_0}) \leq \mu$, then for M independent samples, sampling from the distribution $\mathcal{P}$ of $(k-1)x_1 + kx_2$ will be λ-bit secure, if $\Delta(\mathcal{P}||\mathcal{D}_\sigma) \leq 2\epsilon + 2\mu \leq \sqrt{1/(4\lambda \cdot M)}$.

Lemma 2. Let $x_1, x_2 \hookleftarrow \mathcal{D}'_{\sigma_0}$, with $\sigma_0 = \sigma/\sqrt{1 + k^2}$ for some $\sigma \in \mathbb{R}^+$ and $k \geq 2$. If:

$$\sigma_0 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi},$$
$$\sqrt{\frac{1}{\sigma_0^{-2} + (k\sigma_0)^{-2}}} \geq k\eta_\epsilon(\mathbb{Z})/\sqrt{2\pi},$$

and $\Delta(\mathcal{D}'_{\sigma_0}||\mathcal{D}_{\sigma_0}) \leq \mu$, then for M independent samples, sampling from the distribution $\mathcal{P}$ of $x_1 + kx_2$ will be λ-bit secure, if $\Delta(\mathcal{P}||\mathcal{D}_\sigma) \leq 4\epsilon + 2\mu \leq \sqrt{1/(4\lambda \cdot M)}$.

Proof: We show that for distributions $\mathcal{P}$ and $\mathcal{Q}$, and M independent samples, sampling from $\mathcal{P}$ will be λ-bit secure, if $ML(\mathcal{P}||\mathcal{Q}) \leq \sqrt{1/(4\lambda \cdot M)}$. Let $\alpha = 2\lambda$. By combining Theorem 2 and Theorem 8, we get:

$$R_{2\lambda}(\mathcal{P}||\mathcal{Q}) \leq 1 + \lambda \cdot (ML(\mathcal{P}||\mathcal{Q}))^2 \leq 1 + 1/(4M)$$
$$\implies ML(\mathcal{P}||\mathcal{Q}) \leq \sqrt{1/(4\lambda \cdot M)}.$$

Then, let $\sigma_0 = \sigma/\sqrt{(k-1)^2 + k^2}$, $\vec{z} = (k-1, k)$, and $\vec{\sigma} = (\sigma_0, \sigma_0)$ in Theorem 5 to get $\Delta(\mathcal{P}||\mathcal{D}_\sigma) \leq 2\epsilon + 2\mu$. Let $\sigma_0 = \sigma/\sqrt{1 + k^2}$, $\sigma_1 = \sigma_0$, and $\sigma_2 = k\sigma_0$ in Theorem 6, we get $\Delta(\mathcal{P}||\mathcal{D}_\sigma) \leq 4\epsilon + 2\mu$. We replace ML with Δ in both Theorem 5 and Theorem 6, then get Lemma 1 and Lemma 2, respectively. □

Since the constraint for σ_0 in Lemma 2 is looser than in Lemma 1 (about $\sqrt{2}$ times), but σ_0 shrinks faster in Lemma

TABLE 1
Total Relative Errors for Different Number of Convolution Levels.

$\vec{z}$	1 Level	2 Levels	3 Levels
$(k-1,k)$	$2\epsilon+2\Delta$	$6\epsilon+4\Delta$	$14\epsilon+8\Delta$
$(1,k)$	$4\epsilon+2\Delta$	$12\epsilon+4\Delta$	$28\epsilon+8\Delta$
Mixed-k	$4\epsilon+2\Delta$	$10\epsilon+4\Delta$	$22\epsilon+8\Delta$

TABLE 2
Convolution Parameters for $\sigma \approx 215$.

Method	l	σ_0	$\vec{z_i}$
KLD [14]	1	19.53	$\vec{z_1}=(1,11)$
KLD [16], [17]	2	6.18	$\vec{z_1}=(1,11)$, $\vec{z_2}=(1,3)$
RD $(k-1,k)$	1	17.92	$\vec{z_1}=(8,9)$
RD $(1,k)$	2	5.67	$\vec{z_1}=(1,12)$, $\vec{z_2}=(1,3)$
RD Mixed-k	2	5.67	$\vec{z_1}=(8,9)$, $\vec{z_2}=(1,3)$

1 instead, one can apply both lemmas on different recursion levels. For example, one may adapt Lemma 1 on all the intermediate levels and use Lemma 2 on the bottom level, to achieve possibly smaller base sampler deviation.

The total relative errors for different number of convolution levels are shown in Table 1. Typically, the standard deviations in lattice-based cryptosystems require 3 levels at maximum. Let Δ be the relative error of the base sampler. The "Mixed-k" in Table 1 represents the example we discussed above.

For $\sigma \approx 215$ in the BLISS-I parameter set, the base sampler deviation σ_0 and convolution parameters $\vec{z_i}$ are shown in Table 2 for $i \leq l$, where l is the number of convolution levels. We assume $M = m \cdot q_s$ with $m = 1024$, $q_s = 2^{64}$, $\lambda = 128$, and $\Delta \leq 2^{-53}$. Compared with the KLD-based convolution schemes [14], [16], [17], our RD-based convolution parameter choice lemmas generate smaller base sampler deviations for the same number of convolution levels.

4.5 Performance

We implement the binary sampling scheme (Fig.1) by combining the constant-time CDT base sampler with the FACCT Bernoulli sampler (Fig.8). We choose the tail-cut bound B by Theorem 1, which guarantees that the R_∞ between the tail-cut and the ideal discrete Gaussian is $\leq \exp(1)$ over all $M = m \cdot q_s$ samples, corresponding to a loss of at most $\log_2(\exp(1)) \approx 1.44$ bits of security for the tail-cut samples relative to the ideal discrete Gaussian sampling case. On the other hand, we choose $\Delta_{\mathcal{D}^+_{\sigma_0}}$ and $\Delta_{\mathcal{B}_p}$ by Theorem 2, which guarantees that we lose at most 1 bit of security due to the relative precision errors, respectively. Hence overall our choice of tail-cut and precision parameters ensure that we lose at most $1+1+1.44 = 3.44$ bits of security with respect to the ideal discrete Gaussian sampling over M samples. Since our FACCT Bernoulli sampler is independent of σ, we pick the precision δ_P of the polynomial approximation and bit length l in Fig.8 by using (2) and (3), respectively. We use double precision floating-point, where the mantissa has $\delta_f = 52$ bits, and we fix the parameters in Table 3 for our implementations in the benchmarks.

We employ the full-table access CDT base sampler. We select the parameters in Table 3 such that the base sampler has about 126-bit absolute precision. We store each CDT entry in two 63-bit integers, then the constant-time comparison of $x < y$, where $0 \leq x, y < 2^{63}$, can be performed by a 64-bit signed integer subtraction, since the sign bit of $x - y$ will be 1 when $x < y$. We compute the CDT in reversed order such that $\mathcal{P}(i) = CDT[i] - CDT[i+1]$ for $i \in [0, B]$, where the subtraction only enlarges the relative error by a factor of about σ_0 [14], [22]. For the uniform sampling over the range $[0, k-1]$, we adapt similar techniques as in [42] to reduce the rejection rate. We generate random integers over a larger range $[0, 2^l - 1]$ instead, where $2^l > k$, and then perform the modulo k operations. In addition, we show how to get the polynomial approximation P in our FACCT sampler implementation by using the `sollya` tool in Fig.10, and we verify $\Delta(P||2^x) < 2^{-45.9}$ over the interval $[0, 1]$.

TABLE 3
Parameters for Implementations.

Parameter	Description	Value
M	Number of discrete Gaussian samples	2^{74}
λ	Security level	128
B	Tail-cut bound	9
δ_P	Precision of the polynomial approximation	45
δ_f	Number of bits in the mantissa	52
l	Bit length in Fig.8	19
$\Delta_{\mathcal{D}^+_{\sigma_0}}$	Relative error of the base sampler	2^{-46}
$\Delta_{\mathcal{B}_p}$	Relative error of the Bernoulli sampler	$2^{-44.99}$

For the benchmarks, we select $\sigma \approx \{2^5, 215, 2^{11}, 17900, 2^{17}, 2^{20}\}$, where 215 (approximately $2^{7.7}$) and 17900 (approximately $2^{14.1}$) are the standard deviations from the BLISS-I [1] and the Dilithium-G [19] recommended parameter sets, respectively. We compare the running time of our implementations with the binary sampling scheme from [13] and the countermeasures from [9], [10]. Since the countermeasures did not have a full implementation code available, we simply replace the Bernoulli sampling subroutine in our non-AVX2 reference implementation with the countermeasures. Because the optimal convolution sampling schemes [17], [33] require major refactoring of the bitslicing base sampler for each different σ, we exclude it from this benchmark. We use the AES256 counter mode with hardware AES instructions (AES-NI) to generate the randomness in all the implementations. We use `clang` 8.0.0 to compile our AVX2 implementation, and use `gcc` 9.1.1 to compile all the other implementations, with the compiling options `-O3 -march=native` enabled for both compilers. The benchmark is running on an Intel i7-7700K CPU at 4.2GHz, with the Hyperthreading and the Turbo Boost disabled. We generate $m = 1024$ samples for 1000 times and measure the median number of the consumed CPU cycles. The comparison results are shown in Fig.11. The "Ref" in the following figures and tables represents non-AVX2 reference implementations. We implement the non-AVX2 reference implementations by using the constant-time SSE4 floating-point instructions, which is available on the Intel Nehalem architecture back to 2008 and all subsequent Intel architectures. However, older Intel CPUs such as the Pentium III may not support constant-time hardware floating-point multiplication instructions [26].

```
> guessdegree(1,[0,1],1b-45,1/2^x);
[9;9]
> P=fpminimax(2^x,9,[|1,D...|],[0,1],floating,relative);
> P;
1 + x * (0.693147180561933806686170100874733179807662963867 19
+ x * (0.240226506876527745593108420507633127272129058837 89
+ x * (5.550410984131824709830738129312521778047084808349 6e-2
+ x * (9.618120933175645231871797591338690835982561111450 2e-3
+ x * (1.333387755250109744584174897852335561765357851982 1e-3
+ x * (1.539604321053863805399131159390435641398653388023 4e-4
+ x * (1.535991421946201169828304100573035384513786993920 8e-5
+ x * (1.230394437555541324973693885491687893818379961885 5e-6
+ x * 1.432910037894390942758726138761549151467988849617 54e-7)))))))
> supnorm(P,2^x,[0,1],relative,1b-128);
[1.491806901685506403985743728294477543016355700589 2e-14;
1.491806901685506403985743728294477543020602726225 8424e-14]
```

Fig. 10. The polynomial approximation P in the FACCT sampler implementation.

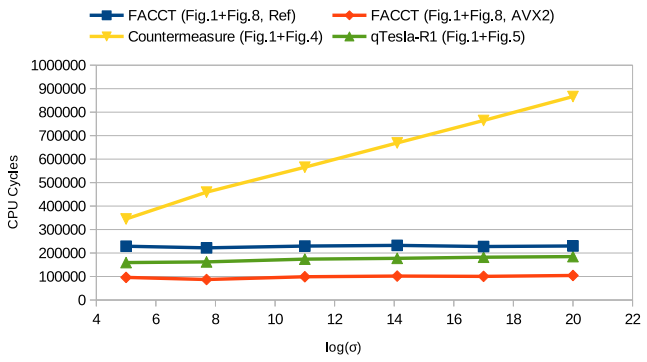

Fig. 11. Comparison of the CPU cycles for different σ.

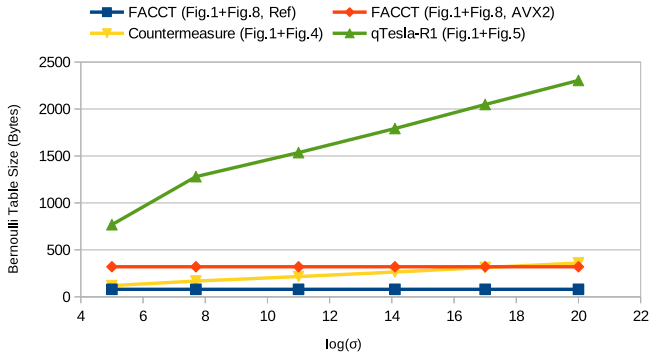

Fig. 12. Comparison of the Bernoulli table size for different σ.

We measure the table size of the Bernoulli sampler by computing the number of table entries times the size of the variable type (in bytes) for each implementation. Since we store vectors instead of single values in our AVX2 implementation, the table size is 4x our non-AVX2 reference implementation. The comparison results are shown in Fig.12.

From Fig.11, compared to the countermeasures, our non-AVX2 reference implementation is 1.5x–3.7x faster, and our AVX2 implementation is 3.5x–8.3x faster, respectively, especially for the larger σ. In addition, our AVX2 implementation is 1.6x–1.8x faster than the qTesla-R1 sampler. Note that our non-AVX2 reference implementation is suboptimal on the running speed, since the constant-time floating-point arithmetic instructions for a single value have similar latencies and throughputs as their SIMD counterparts on the Intel CPUs [35]. Therefore, our optimal AVX2 implementation should be used if running speed is concerned.

From Fig.12, our implementations have much smaller table sizes than the qTesla-R1 sampler (9.6x–28.8x for our non-AVX2 reference implementation and 2.4x–7.2x for our AVX2 implementation), especially for the larger σ. In addition, compared to the countermeasures, our non-AVX2 reference implementation has 1.5x–4.5x smaller table size, and our AVX2 implementation has similar table size, respectively.

From both Fig.11 and Fig.12, we also verify that the efficiency of our implementations is independent of σ.

5 Applications

In this section, we compare the performance of our software implementations with previous implementations from actual cryptosystems.[1]

5.1 Sampling from the BLISS-I Standand Deviation

In this section, we compare the performance of our FACCT sampler implementations with the sampler implementations from qTesla (both R1 [13] and R2 [2]), the bitslicing convolution scheme [17], and previous countermeasures [9], [10], using the BLISS-I parameter set. Since other convolution schemes [14], [16] only have hardware implementations, we only compare with the software implementation of the bitslicing convolution [17].

The BLISS-I parameter set has $k = 254$ and $\sigma \approx 215$. We use the similar benchmark setup as Section 4.5, with $\lambda = 128$, $m = 1024$, and $q_s = 2^{64}$, which gives the same number of samples M as in Table 3. For the bitslicing convolution scheme, we compare with the implementation of 128-bit absolute precision, and we directly use the benchmark

1. The implementation source codes are available at https://gitlab.com/raykzhao/gaussian

TABLE 4
Comparison of the CPU Cycles for Generating $m = 1024$ Samples from $\mathcal{D}_{\mathbb{Z},\sigma}$, with $\sigma \approx 215$.

Scheme	CPU cycles (Ref)	CPU cycles (AVX2)
qTesla-R1 [13]	162215	–
qTesla-R2 [2]	2531610	–
Bitslicing [17]	$\approx$ 532800	$\approx$ 254708
Countermeasure [9], [10]	459297	–
FACCT	221991	87192

TABLE 5
Comparison of the Memory Consumptions for $\sigma \approx 215$.

Scheme	Table	Code	Total
qTesla-R1 [13]	192+1280	597	2069
qTesla-R2 [2]	90816	961	91777
Bitslicing [17]	–	$\approx$ 98816	$\approx$ 98816
Countermeasure [9], [10]	144+168	440	752
FACCT (Ref)	144+80	659	883
FACCT (AVX2)	576+320	1275	2171

script[2] from the authors to measure the number of the CPU cycles of generating 64 base samples, and scale the result up to $4m = 4096$ base samples. We also scale this number by the same factor as in [17] to retrieve the AVX2 result. The CPU cycles are shown in Table 4.

To measure the memory consumption of each implementation, we compute the table sizes for both the base samplers and the Bernoulli samplers by using similar approaches as in Section 4.5. Since the bitslicing approach does not require a table, but has a rather large code size [17], for a fair comparison, we also measure the assembly code size (in bytes) of the sampling functions. We compile the source codes by using the compiling options `-Os -march=native` to generate more compact assembly code, and use the `objdump` command to perform the disassembly. The memory consumption comparison results are shown in Table 5. The "Table" represents the total table size, and for binary sampling variants, the results are in the form of "base sampler table size+Bernoulli sampler table size". The "Code" represents the code size, and the "Total" represents the sum of the table size and the code size. All the numbers in Table 5 are in bytes.

From Table 4, in addition to the results from Fig.11, our implementations significantly outperform the bitslicing convolution scheme (2.4x for the reference implementation and 2.9x for the AVX2 implementation). Our implementations are also significantly faster than the qTesla-R2 sampling algorithm for larger $\sigma \approx 215$ (11.4x for the reference implementation and 29.0x for the AVX2 implementation).

From Table 5, in addition to the results from Fig.12, our non-AVX2 reference implementation consumes 2.3x smaller memory space than the qTesla-R1 sampler, and has similar memory consumption compared to the countermeasures, respectively. Our AVX2 implementation has similar memory consumption compared to the qTesla-R1 sampler. However, for larger σ, as shown in Fig.12, the qTesla-R1 sampler will consume significantly more memory space to store the Bernoulli table, while our implementations maintain similar memory consumptions. Both of our implementations consume much smaller memory space than the bitslicing convolution scheme (111.9x for the non-AVX2 reference implementation and 45.5x for the AVX2 implementation). In addition, our implementations have much lower memory consumptions compared to the qTesla-R2 sampler for larger $\sigma \approx 215$ (103.9x for the non-AVX2 reference implementation and 42.2x for the AVX2 implementation).

2. https://github.com/Angshumank/const_gauss

TABLE 6
Comparison of the CPU Cycles for qTesla-R2 (AVX2) Keygen.

Scheme	Orig. (cSHAKE) [2]	Orig. (AES-NI)	FACCT
I	1093917	1009155	402022
III	2875728	2419416	1039426
V	14352751	11607570	4007417

TABLE 7
Signing Speed Comparison for Falcon.

N	Orig. (sig/s)	Our Impl. (sig/s)
256	17844.457	16375.575
384	10232.885	9561.668
512	8781.839	8076.828
768	5281.893	4933.550
1024	4443.585	4086.705

5.2 qTesla

To test the running speed of our sampler in a cryptosystem, we replace the sampler in the AVX2 implementation of qTesla-R2 with our FACCT AVX2. Since the cSHAKE software random generator is much slower than the AES-NI, we measure the performance after replacing the random generator of the sampler with the AES-NI in the implementations. The CPU cycles measured by the benchmark script from qTesla-R2 are shown in Table 6. The qTesla-R2 Keygen with our AVX2 sampler (AES-NI) is 2.3x–2.8x faster than the original implementations (modified to use AES-NI instead of cSHAKE). Note that the standard deviations in qTesla-R2 ($\sigma \approx$ 10.2–22.93) is smaller than the deviations in previous benchmarks. Therefore, our implementation maintains good performance even for smaller σ.

5.3 Falcon

To test the performance of our proposed constant-time $\exp(x)$ implementation in Section 4.1, we replace the $\exp(x)$ in Falcon with our non-AVX2 reference implementation. Since the $\exp(x)$ is used when performing the rejection sampling from the arbitrary-centered discrete Gaussian in the signing, we measure the signing speed by using the benchmark script from Falcon. The results are shown in Table 7. Our constant-time $\exp(x)$ reference implementation only adds very slight overhead (6.5%–8.2%) to the signing (However, the rejection rate of sampling may still be secret dependent).

6 Conclusion

In conclusion, we present fast, compact, and constant-time (FACCT) centered discrete Gaussian sampler over integers, by implementing the Bernoulli sampler in the binary sampling scheme with a constant-time $\exp(x)$ polynomial approximation. Our implementation is faster than previous

countermeasures [9], [10], more compact than the qTesla samplers (both R1 [13] and R2 [2]), and outperforms the bitslicing convolution scheme [17] in both timing and memory consumption. Our implementation techniques are also independent of the standard deviation, and have good flexibility and performance in various applications. In addition, we show the smaller base sampler deviations for the convolution schemes by adapting the Rényi divergence.

Acknowledgments

Ron Steinfeld was supported in part by ARC Discovery Project grant DP180102199.

References

[1] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *CRYPTO (1)*, ser. Lecture Notes in Computer Science, vol. 8042. Springer, 2013, pp. 40–56.

[2] E. Alkim, P. S. L. M. Barreto, N. Bindel, P. Longa, and J. E. Ricardini, "The lattice-based digital signature scheme qtesla," *IACR Cryptology ePrint Archive*, vol. 2019, p. 85, 2019.

[3] L. T. Phong, T. Hayashi, Y. Aono, and S. Moriai, "Lotus: Algorithm specifications and supporting documentation," https://www2.nict.go.jp/security/lotus/index.html, 2017, accessed: 2019-06-27.

[4] M. Seo, S. Kim, D. H. Lee, and J. H. Park, "Emblem: (ring) lwe-based key encapsulation with a new multi-bit encoding method," https://pqc-emblem.org/, 2018, accessed: 2019-06-27.

[5] J. Lee, D. Kim, H. Lee, Y. Lee, and J. H. Cheon, "Rlizard: Post-quantum key encapsulation mechanism for iot devices," *IEEE Access*, vol. 7, pp. 2080–2091, 2019.

[6] D. E. Knuth, *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[7] L. Devroye, *Non-Uniform Random Variate Generation*. New York, NY, USA: Springer-Verlag, 1986.

[8] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme," in *CHES*, ser. Lecture Notes in Computer Science, vol. 9813. Springer, 2016, pp. 323–345.

[9] P. Pessl, L. G. Bruinderink, and Y. Yarom, "To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures," in *CCS*. ACM, 2017, pp. 1843–1855.

[10] T. Espitau, P. Fouque, B. Gérard, and M. Tibouchi, "Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers," in *CCS*. ACM, 2017, pp. 1857–1874.

[11] J. Bootle, C. Delaplace, T. Espitau, P. Fouque, and M. Tibouchi, "LWE without modular reduction and improved side-channel attacks against BLISS," in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 11272. Springer, 2018, pp. 494–524.

[12] 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 *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2015, pp. 553–570.

[13] N. Bindel, S. Akleylek, E. Alkim, P. S. L. M. Barreto, J. Buchmann, E. Eaton, G. Gutoski, J. Kramer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon, "Submission to NIST's post-quantum project: lattice-based digital signature scheme qTESLA," https://qtesla.org/, 2017, accessed: 2018-11-03.

[14] T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware," in *CHES*, ser. Lecture Notes in Computer Science, vol. 8731. Springer, 2014, pp. 353–370.

[15] D. Micciancio and M. Walter, "Gaussian sampling over the integers: Efficient, generic, constant-time," in *CRYPTO (2)*, ser. Lecture Notes in Computer Science, vol. 10402. Springer, 2017, pp. 455–485.

[16] A. Khalid, J. Howe, C. Rafferty, F. Regazzoni, and M. O'Neill, "Compact, scalable, and efficient discrete gaussian samplers for lattice-based cryptography," in *ISCAS*. IEEE, 2018, pp. 1–5.

[17] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Constant-time discrete gaussian sampling," *IEEE Trans. Computers*, vol. 67, no. 11, pp. 1561–1571, 2018.

[18] M. F. Esgin, R. Steinfeld, A. Sakzad, J. K. Liu, and D. Liu, "Short lattice-based one-out-of-many proofs and applications to ring signatures," *IACR Cryptology ePrint Archive*, vol. 2018, p. 773, 2018.

[19] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS - dilithium: Digital signatures from module lattices," *IACR Cryptology ePrint Archive*, vol. 2017, p. 633, 2017.

[20] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld, "Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance," in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 9452. Springer, 2015, pp. 3–24.

[21] T. Prest, "Sharper bounds in lattice-based cryptography using the rényi divergence," in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 10624. Springer, 2017, pp. 347–374.

[22] C. A. Melchor and T. Ricosset, "Cdt-based gaussian sampling: From multi to double precision," *IEEE Trans. Computers*, vol. 67, no. 11, pp. 1610–1621, 2018.

[23] N. C. Dwarakanath and S. D. Galbraith, "Sampling from discrete gaussians for lattice-based cryptography on a constrained device," *Appl. Algebra Eng. Commun. Comput.*, vol. 25, no. 3, pp. 159–180, 2014.

[24] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," https://falcon-sign.info/, 2017, accessed: 2018-10-31.

[25] M. Walter, "Sampling the integers with low relative error," *IACR Cryptology ePrint Archive*, vol. 2019, p. 68, 2019.

[26] G. Barthe, S. Belaïd, T. Espitau, P. Fouque, M. Rossi, and M. Tibouchi, "GALACTICS: gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited," *IACR Cryptology ePrint Archive*, vol. 2019, p. 511, 2019.

[27] C. Peikert, "An efficient and parallel gaussian sampler for lattices," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 6223. Springer, 2010, pp. 80–97.

[28] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Compact and side channel secure discrete gaussian sampling," *IACR Cryptology ePrint Archive*, vol. 2014, p. 591, 2014.

[29] M. O. Saarinen, "Arithmetic coding and blinding countermeasures for ring-lwe," *IACR Cryptology ePrint Archive*, vol. 2016, p. 276, 2016.

[30] P. Pessl, "Analyzing the shuffling side-channel countermeasure for lattice-based signatures," in *INDOCRYPT*, ser. Lecture Notes in Computer Science, vol. 10095, 2016, pp. 153–170.

[31] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill, "On practical discrete gaussian samplers for lattice-based cryptography," *IEEE Trans. Computers*, vol. 67, no. 3, pp. 322–334, 2018.

[32] NIST, "NIST post-quantum competition," http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf, 2016, accessed: 2018-10-31.

[33] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Pushing the speed limit of constant-time discrete gaussian sampling. A case study on the falcon signature scheme," in *DAC*. ACM, 2019, pp. 88:1–88:6.

[34] D. E. Knuth and A. C. Yao, "The complexity of non-uniform random number generation," *Algorithms and Complexity: New Directions and Recent Results*, pp. 357–428, 1976.

[35] Intel, "Intel intrinsics guide," https://software.intel.com/sites/landingpage/IntrinsicsGuide/, accessed: 2018-10-31.

[36] G. Seiler, "Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography," *IACR Cryptology ePrint Archive*, vol. 2018, p. 39, 2018.

[37] J. Muller, N. Brisebarre, F. de Dinechin, C. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[38] S. Chevillard, M. Joldes, and C. Q. Lauter, "Sollya: An environment for the development of numerical codes," in *ICMS*, ser. Lecture Notes in Computer Science, vol. 6327. Springer, 2010, pp. 28–31.

[39] S. Chevillard, C. Lauter, and M. Joldes, "Users' manual for the sollya tool," https://gforge.inria.fr/frs/download.php/file/37750/sollya.pdf, accessed: 2018-11-19.

[40] N. Brisebarre and S. Chevillard, "Efficient polynomial l-approximations," in *IEEE Symposium on Computer Arithmetic*. IEEE Computer Society, 2007, pp. 169–176.

[41] Y. Du, B. Wei, and H. Zhang, "A rejection sampling algorithm for off-centered discrete gaussian distributions over the integers,"

SCIENCE CHINA Information Sciences, vol. 62, no. 3, pp. 39 103:1–39 103:3, 2019.

[42] R. Steinfeld, A. Sakzad, and R. K. Zhao, "Titanium: Proposal for a nist post-quantum public-key encryption and kem standard specifications document version 1.1," http://users.monash.edu.au/~rste/Titanium_v11.pdf, submitted to NIST Post-Quantum Competition. Accessed: 2019-01-08.

Raymond K. Zhao is a current Ph.D. candidate at the Faculty of Information Technology, Monash University, Australia. He received his B.Eng. degree in Computer Science and Technology from Zhejiang University, China, in 2015, and his Master degree in Network and Security from Monash University, Australia, in 2017. His main research interests include efficient and side-channel resistant implementation techniques on lattice-based cryptography and its applications.

Ron Steinfeld (S'99-M'04) is a Senior Lecturer at the Clayton School of IT, Faculty of Information Technology, Monash University, Australia since 2015. He received his B.Sc. degree in Mathematics and Physics in 1998, his B.E. Hons (First Class) degree in Electrical and Computer Systems Engineering in 2000, and his Ph.D. degree in Computer Science in 2003, all from Monash University, Australia. From 2003 to 2006 he was a postdoctoral research fellow in cryptography and information security at Macquarie University, Australia. From 2007 to 2009 he was a Macquarie University Research Fellow in cryptography and information security. From 2009 to 2014 he was an ARC Australian Research Fellow in cryptography and information security at Macquarie University (until 2012) and then at Monash University (2012-2014). His main research interests include the design and analysis of cryptographic algorithms and cybersecurity protocols.

Amin Sakzad (M'12) received his Ph.D. degree in applied mathematics from the Amirkabir University of Technology (Tehran Polytechnique), Tehran, Iran, in 2011. He was a research assistant and a lecturer at Carleton University, Ottawa, ON, Canada, in 2010. He was a Research visitor and a lecturer at the Amirkabir University of Technology in 2011. He was a research fellow with the Software Defined Telecommunications (SDT) Laboratory in the Department of Electrical and Computer Systems Engineering (ECSE) at Monash University, Melbourne, Australia from 2012-2015. Starting from 2016, he held a posdoctoral research fellowship position at the Faculty of Information Technology (FIT), Monash University, Melbourne, Australia. As of May 2017, he has been appointed as a lecturer at FIT, Monash University, Melbourne, Australia. His research interests include Euclidean lattices, lattice-based cryptography, and wireless network coding.