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

Raymond K. Zhao, Ron Steinfeld and Amin Sakzad

Faculty of Information Technology, Monash University,
{raymond.zhao,ron.steinfeld,amin.sakzad}@monash.edu

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 Renyi 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.

Keywords: Lattice-based crypto · Discrete Gaussian sampling · Constant-time · Implementation · Efficiency

1 Introduction

1.1 Background

The discrete Gaussian sampler over integers is an important tool in implementing lattice-based cryptosystems, especially lattice-based signature schemes [DDLL13, BAA+17]. However, typical discrete Gaussian sampling implementations usually suffer from side-channel vulnerabilities. The straightforward Knuth-Yao [Knu98] or binary-search Cumulative Distribution Table (CDT) [Dev86] approaches are not constant-time, due to the branch conditions. For the binary sampling algorithm from the BLISS signature [DDLL13], the non-constant time implementation recently became the target of several side-channel attacks [BHLY16, PBY17, EFGT17, BDE+18] 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 [BCNS15], 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 [PBY17, EFGT17, BAA+17], and the convolution scheme [PDG14, MW17], both have drawbacks: For the first binary sampling expander approach, existing constant-time countermeasures either introduce significant overhead in the running speed [PBY17, EFGT17], or add large look-up tables [BAA+17]. 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 [PDG14, KHR+18], 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 [KRR+18] 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 [ESS+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 [DLL+17] moved away from using the discrete Gaussian distribution, at the expense of larger signature size.

1.2 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 [BAA+17], or combining many Bernoulli samples [PBY17, EFGT17], the $\exp(x)$ function polynomial approximation techniques with a carefully chosen precision can achieve faster and more compact constant-time implementations of the binary sampling expander. To minimise the required polynomial approximation precision, we show how to apply the Renyi divergence analysis technique to the binary sampling algorithm. Previous works on the Renyi divergence used a different order [BLL+15], only applied this technique to the rejection in the BLISS signing algorithm [Pre17], or applied to a different sampling method [MR18]. As opposed to [DG14], 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 [BAA+17, PFH+17].

- As an additional independent contribution, we show how to adapt the Renyi divergence analysis technique to the convolution sampling algorithm, and achieve smaller σ_0 for the base sampler, compared to the existing Kullback-Leibler Divergence (KLD) based algorithms [PDG14, KHR+18].

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 $\Pr(X=0)=1-p$). 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 $\rho_{1/(s\cdot\sqrt{2\pi})}(\Lambda^* \setminus \{0\}) \leq \epsilon$, where Λ^* is the dual lattice of Λ: $\Lambda^* = \{\vec{w} \in \mathbb{R}^n : \forall \vec{x} \in \Lambda, \vec{x}\cdot\vec{w} \in \mathbb{Z}\}$ [Pei10]. An upper bound on $\eta_\epsilon(\mathbb{Z})$ is given by [Pei10]: $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2+2/\epsilon)/\pi}$.

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 [PDG14])**.** 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 (Renyi Divergence [BLL+15, Pre17])**.** For two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) \subseteq \mathrm{Supp}(\mathcal{Q})$, the Renyi 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 [MW17])**.** 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 [BLL+15], 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 [Pre17], 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:*

$$R_\alpha(\mathcal{P}||\mathcal{Q}) \leq \left(1 + \frac{\alpha(\alpha-1)\cdot\left(\Delta\left(\mathcal{P}||\mathcal{Q}\right)\right)^2}{2\left(1-\Delta\left(\mathcal{P}||\mathcal{Q}\right)\right)^{\alpha+1}}\right)^{\frac{1}{\alpha-1}} \underset{\Delta(\mathcal{P}||\mathcal{Q})\to 0}{\sim} 1 + \frac{\alpha\cdot\left(\Delta\left(\mathcal{P}||\mathcal{Q}\right)\right)^2}{2}.$$

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 [DDLL13], and the convolution methods [PDG14, MW17].

3.1 Binary Sampling Method

The original binary sampling method was proposed by [DDLL13] 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 from the base sampler: $x \hookleftarrow \mathcal{D}_{\sigma_0}^+$, 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 [DDLL13], 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:*

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

The rejection framework of the binary sampling algorithm is shown in Algorithm 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 Algorithm 2 and Algorithm 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 [BHLY16], or timing and power [EFGT17], 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.

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

Random Shuffle. One commonly used heuristic countermeasure is performing the Fisher-Yates random shuffle (or Knuth shuffle) [Knu98], to mask the relation between the retrieved side-channel information of the samples and the secret, after performing non-constant time sampling schemes [RRVV14, Saa16]. 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 [Pes16] that an attacker only requires marginally larger yet still practical number of samples to rearrange the coordinates and "undo" the shuffle.

Algorithm 1 Binary Sampling Scheme [DDLL13]

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

Algorithm 2 Base Sampler from BLISS [DDLL13]

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

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 [BCNS15]. A recent work [HKR+18] 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 [KRR+18]. On the other hand, for the table-based Bernoulli sampler, several works [BHLY16, PBY17, EFGT17] suggested the countermeasure of removing the branches and performing full-table access (see Algorithm 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 [NIS16], qTesla [BAA+17], 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 Algorithm 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.

Algorithm 3 Bernoulli Sampler from BLISS [DDLL13]

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 $p_i = \exp\left(-2^i/2\sigma^2\right)$ for $i < l$.
Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/2\sigma^2)$.

```
function BernoulliSampler(t)
    for i = l − 1 downto 0 do
        if t_i = 1 then
            Sample a ↩ B_{p_i}.
            if a = 0 then
                return 0.
            end if
        end if
    end for
    return 1.
end function
```

Algorithm 4 Constant-time Bernoulli Sampler [PBY17, EFGT17]

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 $p_i = \exp\left(-2^i/2\sigma^2\right)$ 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 ↩ B_{p_i}.
        Set r = r · (1 − t_i + a t_i).
    end for
    return r.
end function
```

3.3 Convolution Methods

Previous works [PDG14, KHR+18] suggested applying the following KLD-based convolution theorem to construct discrete Gaussian sampling algorithms:

Theorem 4 (KLD-based Convolution Theorem, Adapted from [PDG14], 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 Algorithm 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 $\mathcal{D}_{\sigma_2}$, a recent work [KRR+18] adapted the bitslicing method to implement the Knuth-Yao algorithm [KY76] more efficiently in constant-time, compared to the previous full-table access CDT approach.

Meanwhile, a recent work [MW17] proposed the following max-log based convolution theorems:

Algorithm 5 Bernoulli Sampler with Constant Number of Iterations [BAA+17]

Input: Integer $t = y(y + 2kx)$ with $0 \leq t < 2^{15}$, where $x \hookleftarrow \mathcal{D}^{+}_{\sigma_0}$ and $y \hookleftarrow \mathcal{U}(\{0, \ldots, k-1\})$. Pre-computed Bernoulli table entries $\mathcal{B}_{i,j} = \exp\left(-2^{(j \cdot 32^i)}/2\sigma^2\right)$, where $i, j \in \mathbb{Z}^+$, $0 \leq i < 3$, $0 \leq j < 32$. 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 ↩ U({0,1}^62).
    Let c = 2^62.
    Let s = t.
    for i = 0 to 2 do
        Set c = c · B_{i, s mod 32}.
        Set s = s/32.
    end for
    if r ≥ ⌊c⌋ then
        return 0.
    else
        return 1.
    end if
end function
```

Algorithm 6 KLD-based Convolution Sampling Scheme [PDG14, KHR+18]

Output: A sample from $\mathcal{D}_\sigma$, where $\sigma \approx 215$.

```
function ConvolutionSampler
    Sample x_1, x_2, x_3, x_4 ↩ D_{σ_0}, where σ_0 ≈ 6.18.
    Let y = (x_1 + 3x_2) + 11 · (x_3 + 3x_4).
    return y.
end function
```

Theorem 5 (Adapted from [MW17], Cor. 4.1)**.** *Let $\vec{z} = (z_1, \ldots, z_n) \in \mathbb{Z}^n$ be a nonzero vector with $\gcd(z_1, \ldots, z_n) = 1$, and $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n$ 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 [MW17], 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 [Pre17]. Falcon [PFH+17], 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 an $\exp(x)$ implementation similar to the C standard library (see Algorithm 7). However, the floating-point division instructions on the Intel CPUs have various latency and throughput [Int]. Furthermore, the compiler may replace the division operation with its own arithmetic library routine, which may not be constant-time

[Sei18]. Therefore, the division arithmetic should be generally avoided in constant-time implementation.

Algorithm 7 $\exp(x)$ [PFH+17]

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

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 [Int]. 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 $t \geq 0$.

2. Second, we adapt the method from [MBdD+10]. 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 $2^{\lfloor a \rfloor}$ by changing the exponent of a double precision variable. To approximate 2^z, we use the `sollya` tool [CJL10] 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 [CLJ], we use the following two functions to get such a polynomial:

guessdegree. The guessdegree(f, I, δ, ω) 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:

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

fpminimax. The fpminimax$(f, n, L, I, \text{floating}, \text{relative})$ function performs the heuristic from [BC07] 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 $c_0 = 1$.

supnorm. The 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.

Our constant-time Bernoulli sampler adapting the $\exp(x)$ approximation approach above is shown in Algorithm 8. Here we analyse the relative error of Algorithm 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:

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

Since we represent $f = P(z) \cdot 2^{\lfloor a \rfloor}$ as $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$ with $(\delta_f + 1)$-bit precision, we have:

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

Then, the relative error between $\mathcal{P}_{\text{FACCT}}$ and $\mathcal{P}_{\text{IDEAL}}$ is:

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

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

and Δ, we have:

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

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

$$l \geq 2B + 1 \tag{3}$$

Algorithm 8 FACCT Bernoulli Sampler

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 for $0 \leq z < 1$, with δ_P-bit precision. Assume a floating-point value $f \in (0, 1]$ with $(\delta_f + 1)$-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 $exponent \in \mathbb{Z}^-$. 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 $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$.
- 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** $\left(r_m < mantissa + 2^{\delta_f}$ **and** $r_e < 2^{l + exponent + 1}\right)$ **or** $f = 1.0$ **then**
  - **return** 1.
- **else**
  - **return** 0.
- **end if**

end function

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 the Intel intrinsics. This also 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 [BHLY16, PBY17, EFGT17, BAA+17], 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 11 coefficients for double precision in our implementation. 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 the 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.

4.2 Concrete Renyi Divergence Based Convolution Sampling

Previous works [Pre17] only implied the potentially tighter parameters for the convolution theorem based samplers by adapting the Renyi divergence. In this section, we discuss the concrete parameter choice for the RD-based convolution sampling scheme.

Theorem 7 (Adapted from [Pre17], Lemma 4)**.** *For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that* $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$*, we have:*

$$R_\alpha(\mathcal{P}||\mathcal{Q}) \leq 1 + \frac{\alpha \cdot (ML\,(\mathcal{P}||\mathcal{Q}))^2}{2},$$

when $ML(\mathcal{P}||\mathcal{Q}) \to 0$.

Recall that $ML(\mathcal{P}||\mathcal{Q}) \approx \Delta(\mathcal{P}||\mathcal{Q})$ when $\Delta(\mathcal{P}||\mathcal{Q}) \to 0$ (Lemma 4.2, [MW17]). Also, in the convolution sampler adapting Theorem 5, typically $\vec{z} = (k-1, k)$ for some $k \geq 4$ [MW17]. 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 $\sigma \in \mathbb{R}^+$, $k \geq 4$. If $\sigma_0 \geq k\eta_\epsilon(\mathbb{Z})/\sqrt{\pi}$, and $\Delta\left(\mathcal{D}'_{\sigma_0}||\mathcal{D}_{\sigma_0}\right) \leq \mu$, then for M independent samples, sampling from the distribution $\mathcal{P}$ of $(k-1)x_1 + kx_2$ will be λ-bit secure, if $\Delta(\mathcal{P}||\mathcal{D}_\sigma) \leq 2\epsilon + 2\mu \leq \sqrt{1/\left(4\lambda \cdot M\right)}$.*

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

Proof. We show that for distributions $\mathcal{P}$, $\mathcal{Q}$, and M independent samples, sampling from $\mathcal{P}$ will be λ-bit secure, if $ML(\mathcal{P}||\mathcal{Q}) \leq \sqrt{1/\left(4\lambda \cdot M\right)}$. Let $\alpha = 2\lambda$. By combining Theorem 2 and Theorem 7, we get:

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

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 1, 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 $m = 1024$ and $q_s = 2^{64}$, $\lambda = 128$, and $\Delta \leq 2^{-53}$. Compared with the KLD-based convolution schemes [PDG14, KHR+18, KRR+18], our RD-based convolution parameter choice lemmas generate smaller base sampler deviations for the same number of convolution levels.

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 [PDG14]	1	19.53	$\vec{z_1} = (1, 11)$
KLD [KHR+18, KRR+18]	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	$\vec{z_1} = (8, 9)$, $\vec{z_2} = (1, 3)$

4.3 Performance

We implement the binary sampling scheme (Algorithm 1) by combining the constant-time CDT base sampler with the FACCT Bernoulli sampler (Algorithm 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 Algorithm 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, and the sign bit of $x - y$ will represent whether x is smaller than 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 [PDG14, MR18]. For the uniform sampling over the range $[0, k-1]$, we adapt similar techniques as in [SSZ] 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

Table 3: Parameters for benchmarks

Parameter	Description	Value
m	Dimension of the lattice	1024
q_s	Number of queries	2^{64}
λ	Security level	128
B	Tail-cut bound	9
δ_P	Precision of the polynomial approximation (bits)	45
δ_f	Number of bits in the mantissa	52
l	Bit length in Algorithm 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	$2^{-44.99}$

our FACCT sampler implementation by using the `sollya` tool in Figure 1, and we verify $\Delta(P||2^x) < 2^{-45.9}$ over the interval $[0, 1]$.

```
> 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 *
1.432910037894390942758726138761549151467988496175 4e-7))))))))
> supnorm(P,2^x,[0,1],relative,1b-128);
[1.491806901685506403985743728294477543016355700589 2e-14;
1.491806901685506403985743728294477543020602726225 8424e-14]
```

Figure 1: The polynomial approximation P used in the FACCT sampler implementation

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 [DDLL13] and the Dilithium-G [DLL+17] recommended parameter sets, respectively. We compare the running time of our implementations with the binary sampling scheme from [BAA+17], and the countermeasures from [PBY17, EFGT17]. 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 scheme [KRR+18] requires major refactoring of the bitslicing base sampler for each different σ, we exclude it from this benchmark. We use hardware AES instructions to generate the randomness in all the implementations. We use `clang` 7.0.0 to compile our AVX2 implementation, and use `gcc` 7.4.1 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 Figure 2. The "Ref" in the following figures and tables represents non-AVX2 reference implementations.

We measure the table size of the Bernoulli sampler by computing the number of table

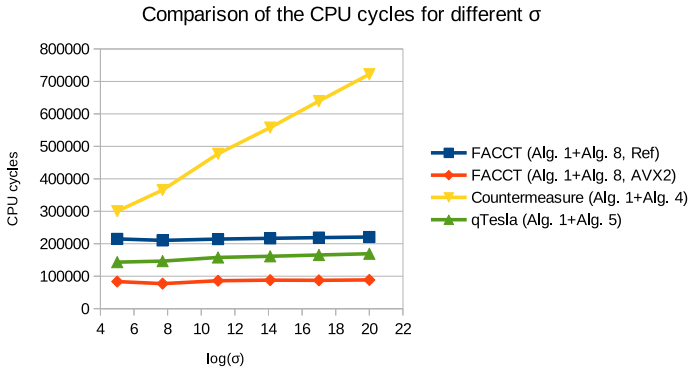

Figure 2: Comparison of the CPU cycles for different σ

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 Figure 3.

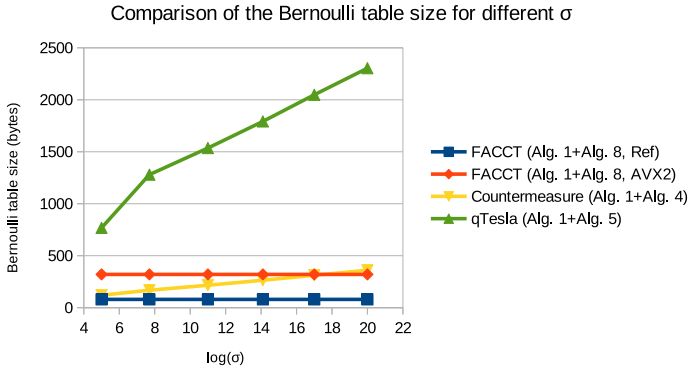

Figure 3: Comparison of the Bernoulli table size for different σ

From Figure 2, compared to the countermeasures, our non-AVX2 reference implementation is 1.3x~3.2x faster, and our AVX2 implementation is 3.5x~8.1x faster, respectively, especially for the larger σ. In addition, our AVX2 implementation is 1.7x~1.9x faster than the qTesla sampler. Note that our non-AVX2 reference implementation is suboptimal on the running speed, since the floating-point arithmetic instructions for a single value have similar latencies and throughputs as their SIMD counterparts on the Intel CPUs [Int]. Therefore, our optimal AVX2 implementation should be used if running speed is concerned.

From Figure 3, our implementations have much smaller table sizes than the qTesla 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 Figure 2 and Figure 3, we also verify that the efficiency of our implementations is independent of σ.

5 Applications

In this section, we show the performance of our implementations in actual cryptosystems.[1]

5.1 Sampling from the BLISS-I Standand Deviation

We show more performance details for $k = 254$ and $\sigma \approx 215$ in the BLISS-I parameter set. We use the similar benchmark setup as Section 4.3, where $\lambda = 128$, $m = 1024$ and $q_s = 2^{64}$. In addition, we include the the convolution scheme with the bitslicing base sampler (128-bit absolute precision) from [KRR+18] in this benchmark. For the convolution scheme, we directly use the benchmark 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 [KRR+18] to retrieve the AVX2 result. The CPU cycles are shown in Table 4.

Table 4: Comparison of the CPU cycles for generating $m = 1024$ samples from $\mathcal{D}_{\mathbb{Z},\sigma}$, where $\sigma \approx 215$

Scheme	CPU cycles (Ref)	CPU cycles (AVX2)
qTesla [BAA+17]	146437	–
Bitslicing [KRR+18]	≈ 521472	≈ 240084
Countermeasure [PBY17, EFGT17]	365423	–
FACCT	210312	77175

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 Section 4.3. Since the bitslicing approach does not require a table, but has a rather large code size [KRR+18], 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 "Base" and "Bernoulli" represent the table size of the base sampler and the Bernoulli sampler, respectively. The "Code" represents the code size, and the "Total" represents the sum of the base sampler table size, the Bernoulli sampler table size, and the code size. All the numbers in Table 5 are in bytes.

Table 5: Comparison of the memory consumptions for $\sigma \approx 215$

Scheme	Base	Bernoulli	Code	Total
qTesla [BAA+17]	192	1280	608	2080
Bitslicing [KRR+18]	–	–	≈ 99744	≈ 99744
Countermeasure [PBY17, EFGT17]	144	168	446	758
FACCT (Ref)	144	80	679	903
FACCT (AVX2)	576	320	1275	2171

From Table 4, in addition to the results from Figure 2, our implementations significantly outperform the bitslicing convolution scheme (2.4x for the reference implementation, and 3.1x for the AVX2 implementation).

From Table 5, in addition to the results from Figure 3, our non-AVX2 reference implementation consumes 2.3x smaller memory space than the qTesla sampler, and has similar memory consumption compared to the countermeasures, respectively. Our AVX2 implementation has similar memory consumption compared to the qTesla sampler. However,

[1] The implementation source codes are available at https://github.com/raykzhao/gaussian
[2] https://github.com/Angshumank/const_gauss

for larger σ, as shown in Figure 3, the qTesla 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 (110.4x for the non-AVX2 reference implementation, and 45.9x for the AVX2 implementation).

5.2 qTesla

To test the running speed of our sampler in a cryptosystem, we replace the sampler in qTesla with our AVX2 implementation. Since the cSHAKE software random generator is much slower than hardware AES instructions, we measure the performance after changing the random generator of the sampler to hardware AES in the implementations. The CPU cycles measured by the benchmark script from qTesla are shown in Table 7. The qTesla Keygen with our AVX2 sampler is 1.2x~1.4x faster than the original implementations (with hardware AES instructions). Note that the standard deviations in qTesla ($\sigma \approx 7.64$~22.93) is smaller than the deviations in previous benchmarks. Therefore, our implementation maintains good performance even for smaller σ.

Table 6: Comparison of the CPU cycles for the qTesla Keygen

Scheme	Orig. (cSHAKE) [BAA+17]	Orig. (AES-NI)	FACCT (AVX2+AES-NI)
I	1138520	480970	389497
III-size	2376550	1062369	726504
III-speed	3452067	1486056	1017370
p-I	6445054	3067462	2090132
p-III	30748522	14025412	10334475

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 (6.9%~7.6%) to the signing (However, the rejection rate of sampling may still be secret dependent).

Table 7: Signing speed comparison for the Falcon

N	Orig. (sig/s)	Our Impl. (sig/s)
256	17326.542	16057.534
384	10188.836	9465.415
512	8548.610	7893.910
768	5251.137	4885.237
1024	4311.670	3993.353

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 [PBY17, EFGT17], more compact than the qTesla sampler [BAA+17], and outperforms the bitslicing convolution scheme [KRR+18] 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 Renyi divergence.

A recent side-channel attack [BDE+18] against the BLISS signature [DDLL13] exploited the $\cosh(x)$ function instead, where the $\exp(x)$ is used as the subroutine in the implementation. For future works, an interesting question is, could our proposed constant-time $\exp(x)$ implementation be applicable as a countermeasure in this scenario? Another open question is, how to adapt our proposed techniques to implement constant-time sampling algorithms over arbitrary-centered discrete Gaussian distributions.

References

[BAA+17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. Submission to NIST's post-quantum project: lattice-based digital signature scheme qTESLA. https://qtesla.org/, 2017. Accessed: 2018-11-03.

[BC07] Nicolas Brisebarre and Sylvain Chevillard. Efficient polynomial l-approximations. In *IEEE Symposium on Computer Arithmetic*, pages 169–176. IEEE Computer Society, 2007.

[BCNS15] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society, 2015.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In *ASIACRYPT (1)*, volume 11272 of *Lecture Notes in Computer Science*, pages 494–524. Springer, 2018.

[BHLY16] 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 *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345. Springer, 2016.

[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 *ASIACRYPT (1)*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015.

[CJL10] Sylvain Chevillard, Mioara Joldes, and Christoph Quirin Lauter. Sollya: An environment for the development of numerical codes. In *ICMS*, volume 6327 of *Lecture Notes in Computer Science*, pages 28–31. Springer, 2010.

[CLJ] Sylvain Chevillard, Christoph Lauter, and Mioara Joldes. Users' manual for the sollya tool. https://gforge.inria.fr/frs/download.php/file/37750/sollya.pdf. Accessed: 2018-11-19.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.

[Dev86] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, USA, 1986.

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

[DLL+17] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - dilithium: Digital signatures from module lattices. *IACR Cryptology ePrint Archive*, 2017:633, 2017.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *CCS*, pages 1857–1874. ACM, 2017.

[ESS+18] Muhammed F. Esgin, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, and Dongxi Liu. Short lattice-based one-out-of-many proofs and applications to ring signatures. *IACR Cryptology ePrint Archive*, 2018:773, 2018.

[HKR+18] James Howe, Ayesha Khalid, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Trans. Computers*, 67(3):322–334, 2018.

[Int] Intel. Intel intrinsics guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide/. Accessed: 2018-10-31.

[KHR+18] Ayesha Khalid, James Howe, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. Compact, scalable, and efficient discrete gaussian samplers for lattice-based cryptography. In *ISCAS*, pages 1–5. IEEE, 2018.

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[KRR+18] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Trans. Computers*, 67(11):1561–1571, 2018.

[KY76] D. E. Knuth and A. C. Yao. The complexity of non-uniform random number generation. *Algorithms and Complexity: New Directions and Recent Results*, pages 357–428, 1976.

[MBdD+10] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[MR18] Carlos Aguilar Melchor and Thomas Ricosset. Cdt-based gaussian sampling: From multi to double precision. *IEEE Trans. Computers*, 67(11):1610–1621, 2018.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In *CRYPTO (2)*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, 2017.

[NIS16] 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.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In *CCS*, pages 1843–1855. ACM, 2017.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2010.

[Pes16] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *INDOCRYPT*, volume 10095 of *Lecture Notes in Computer Science*, pages 153–170, 2016.

[PFH+17] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. https://falcon-sign.info/, 2017. Accessed: 2018-10-31.

[Pre17] Thomas Prest. Sharper bounds in lattice-based cryptography using the rényi divergence. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 347–374. Springer, 2017.

[RRVV14] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Compact and side channel secure discrete gaussian sampling. *IACR Cryptology ePrint Archive*, 2014:591, 2014.

[Saa16] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for ring-lwe. *IACR Cryptology ePrint Archive*, 2016:276, 2016.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography. *IACR Cryptology ePrint Archive*, 2018:39, 2018.

[SSZ] Ron Steinfeld, Amin Sakzad, and Raymond 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.