Fast Discretized Gaussian Sampling and Post-quantum TLS Ciphersuite

Xinwei Gao[1], Lin Li[1(✉)], Jintai Ding[2(✉)], Jiqiang Liu[1], R. V. Saraswathy[2], and Zhe Liu[3]

[1] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation,
Beijing Jiaotong University, Beijing 100044, People's Republic of China
{xinweigao,lilin,jqliu}@bjtu.edu.cn

[2] Department of Mathematical Sciences, University of Cincinnati,
Cincinnati 45219, USA
jintai.ding@gmail.com, rvsaras86@gmail.com

[3] APSIA, Interdisciplinary Centre for Security, Reliability and Trust (SnT),
University of Luxembourg, Luxembourg City, Luxembourg
sduliuzhe@gmail.com

Abstract. LWE/RLWE-based cryptosystems require sampling error term from discrete Gaussian distribution. However, some existing samplers are somehow slow under certain circumstances therefore efficiency of such schemes is restricted. In this paper, we introduce a more efficient discretized Gaussian sampler based on ziggurat sampling algorithm. We also analyze statistical quality of our sampler to prove that it can be adopted in LWE/RLWE-based cryptosystems. Compared with ziggurat-based sampler by Buchmann et al. related samplers by Peikert, Ducas et al. and Knuth-Yao, our sampler achieves more than 2x speedup when standard deviation is large. This can benefit constructions rely on noise flooding (e.g., homomorphic encryption). We also present two applications: First, we use our sampler to optimize the RLWE-based authenticated key exchange (AKE) protocol by Zhang et al. We achieve 1.14x speedup on total runtime of this protocol over major parameter choices. Second, we give practical post-quantum Transport Layer Security (TLS) ciphersuite. Our ciphersuite inherits advantages from TLS and the optimized AKE protocol. Performance of our proof-of-concept implementation is close to TLS v1.2 ciphersuites and one post-quantum TLS construction.

Keywords: Post-quantum cryptography · Lattice · RLWE · Sampling TLS

1 Introduction

1.1 Backgrounds

Various public key algorithms had been proposed and widely deployed in real world since the ground-breaking Diffie-Hellman key exchange protocol [6].

J. K. Liu and P. Samarati (Eds.): ISPEC 2017, LNCS 10701, pp. 551–565, 2017.
https://doi.org/10.1007/978-3-319-72359-4_33

With the advent of quantum computers however, it is believed that most current public key cryptographic constructions are no longer secure while lattice-based algorithms can survive. Best known attacks on current cryptosystems are Shor's algorithm [20] and Grover's algorithm [11]. Shor's algorithm can break most public key algorithms efficiently when practical quantum computers are available. Grovers algorithm can speedup attacks against most symmetric ciphers and hash functions, but they are considered to be relatively secure [3]. Bennett et al. proved that a quantum computer may provide quadratic speedup on brute-force key search [2] and this attack can be defeated by doubling key length. However, increasing key size while remain practical does not work for public key cryptosystems.

During the past years, lattice-based cryptographic primitives had been recognized for their attractive properties, including resistant to quantum attacks, strong provable security and efficiency. Currently, no public algorithms can efficiently solve hard lattice problems. During the past decade, Learning With Errors (LWE) [19] and Ring-LWE (RLWE) [15] underlie as foundation for numerous modern lattice-based cryptosystems. Constructions based on these hard problems enjoy strong provable security and high efficiency. The secret, fresh and random error term e in LWE/RLWE makes both problems very hard to solve when parameters are properly chosen. For common practices, e and secret key s are sampled from discrete Gaussian distribution, therefore efficient sampling algorithm is essential towards practical LWE/RLWE-based cryptographic constructions. However, some papers have pointed out that sampling may take up too much time in practice. Weiden et al. [21] reported that sampling time takes up $> 50\%$ of total runtime when they implement Lyubashevskys signature scheme [14]. In the authenticated key exchange from ideal lattices protocol proposed at EUROCRYPT 2015 [22], they report that sampling operations may take up $> 60\%$ of total runtime. Therefore, design and implement Gaussian sampler with high efficiency and nice statistical quality become a major technical challenge.

1.2 Related Works

Buchmann et al. proposed the first ziggurat-based discrete Gaussian sampler at SAC 2013 in [5]. This work adapts original ziggurat sampling algorithm designed for continuous Gaussian distribution to discrete case. They claimed that when standard deviation σ is large, their sampler outperforms several common sampling methods. Peikert introduced a sampler using cumulated distribution tables (CDT) at CRYPTO 2010 [18]. This sampler has been proven to be extremely efficient when σ is small, but rather inefficient for large σ. Ducas et al. gave a new sampler that has better trade-off between time and memory at CRYPTO 2013 [9]. It does not use precomputed tables and they claim that sampler is efficient even when σ is large. Knuth-Yao algorithm [12] can sample from Gaussian distribution using binary tree search technique. It is efficient but might cost too much memory. There are various constructions (e.g., homomorphic encryption) that require samples from discrete Gaussian distribution with large σ. This technique is known as noise-flooding.

A RLWE-based authenticated key exchange protocol was proposed at EUROCRYPT 2015 [22] (denoted as AKE15). This protocol behaves in HMQV [13] manner and its hardness is directly based on RLWE problem. It is mutual authenticated, proven secure under Bellare-Rogaway model [1] and forward secure. Bos et al. proposed an implementation of RLWE key exchange protocol at IEEE Symposium on Security and Privacy 2015 [4] (denoted as BCNS15) and integration into TLS. Their ciphersuites adopt RSA or ECDSA as signing algorithm which are vulnerable to quantum computers. Moreover, their ciphersuites cannot achieve mutual authentication.

1.3 Contributions

Our contributions are summarized as follows: First, we introduce a much faster discretized ziggurat Gaussian sampler. We discretize original ziggurat sampling algorithm with several improvement techniques to make it more efficient. We prove that the statistical distance between distribution generated by our sampler and discrete Gaussian distribution is smaller than 2^{-80}, therefore it can be used in lattice-based cryptosystems. Performance of our optimized implementation shows that our sampler is more than 2x speedup over [5,9,18] when σ is large. This could benefit constructions that using distributions with large standard deviations to flood small noises (e.g., homomorphic encryption etc.).

Second, we optimize a RLWE-based authenticated key exchange protocol [22]. We replace the sampler for sampling from distribution with large standard deviation in original AKE15 with our efficient discretized Gaussian sampler. We achieve 1.14x speedup on total runtime of this protocol over major parameter choices.

Third, we integrate our optimized AKE implementation into TLS v1.2 as post-quantum TLS ciphersuite. We also present proof-of-concept implementation and benchmark. Our ciphersuite inherits advantages from both AKE15 and TLS v1.2, including mutual authentication, resistant to quantum attacks and forward secrecy. Performance of our ciphersuite is close to standard TLS v1.2 ciphersuites and BCNS15.

1.4 Organization

In Sect. 2, we recall background knowledge. In Sect. 3, we present our efficient discretized ziggurat-based Gaussian sampler, security proofs, implementation, benchmark and comparison with related works. In Sect. 4.1, we show how our sampler optimizes AKE15 and report benchmarks on 6 parameter choices ranging from 80 to 256-bit security. Section 4.2 introduces our post-quantum TLS ciphersuite, implementation, runtime and comparisons with related works. We conclude the paper in Sect. 5.

2 Preliminaries

2.1 Notation

Let ring $R = Z[x]/(x^n + 1)$ and $R_q = Z_q[x]/(x^n + 1)$. Polynomial $x^n + 1$ is n-th cyclotomic polynomial where n is a power of 2. χ is a probability distribution on R_q, $\leftarrow \chi$ denotes sampling according to distribution χ, $\leftarrow_r$ denotes randomly choosing an element from a finite set. A discrete Gaussian distribution over Z with standard deviation $\sigma > 0$ and mean $c \in Z$ is denoted as $D_{Z,\sigma,c}$. If c is 0, we denote $D_{Z,\sigma,c}$ as $D_{Z,\sigma}$. log denotes natural logarithm. Let L be a discrete subset of Z^m. For any vector $c \in R^m$ and any positive parameter $\sigma \in R > 0$, let $\rho_{\sigma,c}(x) = e^{-\frac{\|x-c\|^2}{2\sigma^2}}$ be the Gaussian function on R^m with center c and parameter σ. Denote $\rho_{\sigma,c}(L) = \sum_{x \in L} \rho_{\sigma,c}(x)$ be the discrete integral of $\rho_{\sigma,c}$ over L, and $D_{L,\sigma,c}$ be the discrete Gaussian distribution over L with center c and parameter σ. Specifically, for all $y \in L$, we have $D_{L,\sigma,c}(y) = \frac{\rho_{\sigma,c}(y)}{\rho_{\sigma,c}(L)}$ [7].

2.2 LWE and RLWE

LWE and its ring variant RLWE are hard problems when parameters are properly chosen. The core idea of these two problems is to perturb random linear equations with small noise. Due to perturbation from error terms, it is very hard to distinguish these equations from truly uniform ones. There are quantum [19] and classical reduction [17] between LWE problem in average-case and worst-case hard lattice problems. If there exists a polynomial-time algorithm to solve LWE/RLWE problem, then there exists algorithms to solve hard lattice problems. Hardness of LWE/RLWE serves as the solid foundation to numerous cryptographic schemes. In practice, RLWE-based schemes are more preferable than LWE-based ones since LWE has an inherent quadratic overhead in computation and communication (large matrix) and this leads to inefficiency. RLWE sample is constructed as polynomial pair (a, b), where $a \in R_q$ is uniformly random, $b = a \cdot s + e \in R_q$, s is small and secret term, e is sampled from discrete Gaussian distribution. Search-RLWE problem is to recover s given many RLWE samples. Decision-RLWE problem is to distinguish b from uniform random. There are similar variants for search-LWE and decision-LWE therefore we ignore details. Cryptographic constructions based on RLWE (e.g., public key encryption, signature, key exchange, homomorphic encryption etc.) can be made truly efficient and practical for real-world deployment.

2.3 Statistical Distance

Since discrete Gaussian distribution has infinitely long tail and high precision for the probabilities of sampled points, it is impossible to generate a truly discrete Gaussian distribution within finite computations. Therefore, it is required that the statistical distance between distribution generated by sampler and discrete Gaussian distribution to be very small.

Statistical distance is defined as follows: If X and Y are two random variables corresponding to given distributions on L, the statistical difference is defined as:

$$\Delta(X, Y) = \frac{1}{2} \sum_{x \in L} |\Pr(X = x) - \Pr(Y = x)| \tag{1}$$

If the statistical distance between two distributions is very small (e.g., $< 2^{-80}$), the difference between these two distributions is negligible.

3 Faster Discretized Gaussian Sampler

Generally, secret key s and error term e of LWE/RLWE-based schemes are sampled from discrete Gaussian distribution. Sampling takes up large portion of runtime in implementation, therefore efficiency of sampling algorithm is very crucial. Ziggurat sampling algorithm [16] can sample from Gaussian distribution very efficiently. However, ziggurat algorithm is designed for continuous distribution and lattice-based schemes require discretized version.

Our sampler is discretized version of [16] and we improve efficiency of our sampler by eliminating computations in sampling operations. We prove that our sampler has very close statistical distance to discrete Gaussian distribution, therefore our sampler can be used in LWE/RLWE-based cryptosystems securely. We also introduce optimized implementation. We explain the construction of our sampler, analyze its statistical quality with proofs, present implementation details, benchmark, discussion and comparisons with several samplers in the following sections.

3.1 Ziggurat Gaussian Sampling Algorithm

We recall the ziggurat Gaussian sampling algorithm [16]: Area A encloses the probability density function $\rho_\sigma(x)$ with n rectangles. Rectangles are chosen in a way such that they have equal area. (x_i, y_i) denotes the coordinate of the lower right corner of each rectangle R_i. R_i^l lies within the area of $\rho_\sigma(x)$ and R_i^r is partly covered by $\rho_\sigma(x)$. We first randomly select $i \in [1, n]$ to select one rectangle, then randomly sample x-coordinate inside R_i by choosing $x' \in [0, x_i]$. If $x' \leq x_{i-1}$, x' is accepted and returned, otherwise we sample a value $\gamma \in [y_{i+1}, y_i]$. If $\gamma + y_{i+1} \leq \rho_\sigma(x')$, x' is accepted and returned, otherwise it is rejected and start over again. The probability of sampling a point in these rectangles are equal since they have same size and rectangle is randomly chosen. Marsaglia also suggested an algorithm for tail region: The following procedure is repeated until $2y > x^2$: uniformly sample $a \in (-1, 0) \cup (0, 1)$ and $b \in (0, 1)$, $x = -\frac{1}{r} \log |a|$, $y = -\log b$. If $a > 0$, return $(r + x)$, else return $-(r + x)$.

3.2 Our Fast Discretized Gaussian Sampling and Statistical Quality Analysis

Our sampler is designed directly based on original ziggurat sampling algorithm, which is designed for continuous Gaussian distribution. We discretize it and improve the efficiency of this algorithm with several optimization techniques. The result is our sampler can get samples subjected to discrete Gaussian distribution efficiently with high statistical quality.

We notice that the most expensive part in original ziggurat algorithm is exponential computation since original ziggurat algorithm requires large amount of exponential computations. This computation is directly related to certain σ, therefore it is more inefficient when σ is large. We improve this by sampling from normal continuous Gaussian distribution ($\sigma = 1$, instead of distribution with certain σ), then multiply the sampled value to σ and a randomly generated sign. Finally, we round it to nearest integer to get discretized value. Our approach effectively avoid the inefficiency where plenty of samplers cannot handle large σ efficiently.

We optimize our sampler even further. We use 3 precomputed tables: *ytab*, *ktab* and *wtab* to reduce online computations. Computations on generating these tables are irrelevant from both sampling computation and different standard deviations, since it is a once-for-all computation. Value of precomputed tables are hard-coded in implementation. We can comfortably use same tables when dealing with different standard deviations. Precomputed tables are generated as follows: $ytab = \rho_1(x_i)$ which stores tabulated values for the height of each ziggurat. *ktab* is for quick acceptance check with $ktab_0 = \lfloor 2^{128} \cdot r \cdot \rho_1(r)/v \rfloor$, $ktab_i = \lfloor 2^{128} \cdot (x_{i-1}/x_i) \rfloor$, $r = x_{127} \approx 3.444286476761$, v is the size of each rectangle. *wtab* is for quick value conversion with $wtab_0 = 0.5^{128} \cdot v/\rho_1(r)$ and $wtab_i = 0.5^{128} \cdot x_i$. We note that other samplers may need to generate precomputed tables again when σ changes while our sampler does not.

Pseudocode of our sampler is given in Algorithm 1 (urandom() refers to generate a uniformly distributed 128-bit precision random float number between 0 and 1):

Here we prove that statistical distance between distribution generated by our sampler and discrete Gaussian distribution is very small. We approximate statistical distance between the distribution generated by our sampler and discrete Gaussian distribution to be less than 2^{-80} for $n = 1024$ samples and $\sigma = 869.632$. We utilize a similar approach as [5] since our sampler takes n samples from discrete Gaussian on Z to get discrete Gaussian distribution samples on Z^n. Conclusion still holds for other parameter choices. We first recall two useful lemmas from [10] for our proofs:

Lemma 1. *Let $\sigma > 0$ and $n \in N$ be fixed. Consider distribution $D_{Z^n,\sigma}$. Let $k \in N$ and suppose $c \geq 1$ is such that:*

$$c > \sqrt{1 + 2\log c + 2(k/n)\log 2} \tag{2}$$

Then:

$$\Pr_{v \leftarrow D_{Z^n,\sigma}}(\|v\| > c\sqrt{n}\sigma) < \frac{1}{2^k} \tag{3}$$

Algorithm 1. Fast Discretized Gaussian Sampling

```
Input: $ytab, ktab, wtab, r, \sigma$
Output: Integer distributed according to discrete Gaussian distribution
while true do
    $i \leftarrow_r \{0, \cdots, 127\}, s \leftarrow_r \{-1, 1\}$
    $r \leftarrow_r$urandom()$, j \leftarrow r \cdot 2^{128}, x \leftarrow j \cdot wtab_i$
    if $j < ktab_i$ then
        break
    end if
    if $i < 127$ then
        $y0 \leftarrow ytab_i, y1 \leftarrow ytab_{i+1}$
        $y \leftarrow y1 + (y0 - u1)\cdot$urandom()
    else
        $x \leftarrow r - \log(1-$urandom()$)/r$
        $y \leftarrow e^{-r(x-0.5r)}\cdot$urandom()
    end if
    if $y < e^{-0.5x^2}$ then
        break
    end if
end while
if $s = 1$ then
    return $\lfloor s \cdot \sigma \rceil$
else
    return $-\lfloor s \cdot \sigma \rceil$
end if
```

Next lemma gives us a way to approximate distributions in Z based on the approximation we need for Z^n:

Lemma 2. *Let $\sigma > 0, \epsilon > 0$ be given. Let $k \in N$ and $t > 0$ be such that the tail bound $\Pr(\|v\| > t\sigma)$ as in Lemma 1 is at most $1/2^k$. For $x \in Z$, denote ρ_x as the probability of sampling x from the distribution D_σ. Suppose one has computed approximations $0 \leq p_x \leq Q$ for $x \in Z$, $-t\sigma \leq x \leq t\sigma$ such that:*

$$|p_x - \rho_x| < \epsilon \tag{4}$$

and such that $\sum_{x=-t\sigma}^{t\sigma} = 1$. Let D' be the distribution on $[-t\sigma, t\sigma] \cap Z$ corresponding to the probabilities p_x.

Denote by D'' the distribution on Z^n corresponding to taking n independent samples v_i from D' and forming the vector $v = (v_1, \cdots, v_n)$. Then:

$$\Delta(D'', D_{Z^n,\sigma}) < 2^{-k} + 2nt\sigma\epsilon \tag{5}$$

Let χ_β denote the distribution generated by our sampler, $D_{Z^n,\beta}$ denote the discrete Gaussian distribution on Z^n. We show the approximation for our sampler using parameters from parameter choice I in Table 3. In order to use Lemma 2, we first need to compute the value of c for $k = 81$ and $n = 1024$ in Lemma 1, thus we have $c = 1.242617$ and this gives us tail $t = c\sqrt{n} \approx 40$. Note that this tail cut is much larger than most samplers (e.g., [5] has tail cut $t = 13$). For our sampler, we have $\Delta(\chi, D_{Z^n,\beta}) < 2^{-k} + 2nt\beta\epsilon$. By choosing the precision level to be 128 for the precomputed tables, we can approximate p_x in the lemma, for the tail cut to be close to discrete Gaussian in Z with the error-constant ϵ as 2^{-128}, therefore we have $\Delta(\chi, D_{Z^n,\beta}) < 2^{-81} + 2 \cdot 1024 \cdot 40 \cdot 869.632 \cdot 2^{-128} < 2^{-80}$. The efficiency of rejection procedure is estimated to be 98.78% [16] which contributes to the performance of our sampler.

3.3 Implementation and Runtime

We use MPFR, GMP and NTL library implement our sampler. We set precision to 128-bit to achieve highly accurate computations. We use 128-bit seed and 128-bit random numbers to remain secure against brute-force quantum attacks. Each value in precomputed tables has 40 significant figures. In one execution, a vector with 2048 samples is generated. Each sampled value mod to a 78-bit prime p and stored in a vec_ZZ_p type vector. We test on a Lenovo ThinkCentre M8500t equipped with 3.6 GHz Intel Core i7-4790 processor running Ubuntu 14.04 64-bit version with 3 GB memory. Our implementation is compiled by g++ 4.8.4 with '-O3 -m64' compilation flags and only runs on single core. We report average runtime of 1,000 times execution of our sampler with different standard deviations σ in Table 1:

Table 1. Performance of our sampler

σ	Million samples/s	σ	Million samples/s	σ	Million samples/s
5	2.94	10^6	2.95	10^{12}	2.91
50	3.01	10^7	2.92	10^{13}	2.89
10^2	2.99	10^8	2.93	10^{14}	2.87
10^3	3.02	10^9	2.90	10^{15}	2.88
10^4	2.97	10^{10}	2.88	10^{16}	2.85
10^5	2.99	10^{11}	2.89	10^{17}	2.84

We also use Valgrind to profile memory cost. Our implementation costs maximum of 11.07 MB memory to generate 2048 samples. Each precomputed table consumes nearly 6 KB of memory. Generate three precomputed tables costs 0.173 s but they are computed offline and values are hard-coded in our implementation. In each execution, same precomputed tables are used and they are irrelevant to different standard deviation. We report this timing for completeness.

3.4 Comparisons and Discussions

We present detailed introduction, analysis and comparison with other samplers in [5,9,12,18]. We also test actual performance of these samplers using same test environment, compiler and compilation flags as Sect. 3.3 with various σ.

A ziggurat-based discrete Gaussian sampler was proposed in [5]. Their approach of adapting original ziggurat algorithm to discrete case is different from ours. Compared with their work, our sampler has following improvements and differences:

1. We effectively avoid expensive computation caused by standard deviation. This major contributes to efficiency of our sampler.
   Bottlenecks of their sampler are:
   - More than 50% of total runtime is spent on computing $e^{-x^2/2\sigma^2}$ (x is also related to σ) in constructing each rectangle.
   - Computation in rejection judgement (calculate $e^{-0.5x^2}$ when judging y is smaller than $e^{-0.5x^2}$ or not).
   - Computation in tail region ($y = e^{-r(x-0.5r)}\cdot$urandom(), line 12 of Algorithm 1).

   It is clear that when σ is large, large amount of time is spent on exponential computation. Our sampler avoids this by sampling from normal distribution first and this is much more efficient.
2. We use 3 precomputed tables to store the values required in sampling procedure, compared to only 1 table to store x_i in their implementation. Our sampler can fetch results from these tables directly instead of online computation, therefore the performance is further improved. In our implementation, multiplication, conversion and generating random numbers take up most time. We use their implementation to test their sampler using same environment and precision with various σ. They claimed that their sampler is the fastest when $\sigma = 1.6 \cdot 10^5$. In our test environment, their sampler produces 1.34 million samples/s and 1.23 s to generate precomputed tables, while our sampler produces 2.97 million samples/s with no additional time cost. We fail to test $\sigma > 10^8$ cases since their code crashes.
3. Their implementation needs to compute precomputed tables again when σ is different. This increases total sampling time significantly. Time spent on generating these tables is not even counted when comparing sampling performance in Table 2. If this part is also included, their sampler is much slower. Our sampler can generate precomputed tables within 0.2 s. These tables are hard-coded in implementation and irrelevant with different σ.
4. Their sampler has statistical distance $< 2^{-100}$ at 106-bit precision and it is better than ours. We are able to achieve much faster sampling at the expense of statistical quality to some extent, but statistical quality of our sampler is still good enough to be adopted in LWE/RLWE-based constructions.

At CRYPTO 2010, Peikert gave a very efficient Gaussian sampler (denoted as PKT) using cumulated distribution table (CDT) [18]. We implement it and benchmark shows that PKT is extremely efficient and much faster than all others when $\sigma < 10^6$, but it can be very slow when σ is large, thus it is more preferable to deal with distributions with smaller σ. We did not count time spent on generating precomputed tables in Table 2 when comparing sampling speed.

Ducas et al. presented a sampling algorithm that offered better trade-off between time and memory at CRYPTO 2013 (denoted as DDLL) [9]. It can sample efficiently without using precomputed tables. We implement DDLL and it is faster than all other samplers (except ours) when σ is large, but our sampler is twice as fast when $\sigma > 10^8$. We note that DDLL consumes less memory than our sampler, thus it is more suitable in resource-constrained devices.

Knuth-Yao algorithm (denoted as KY) [12] can sample from Gaussian distribution efficiently. According to [5], their KY implementation outputs nearly 5.8, 4.9, 3.2 and 1.2 million samples/s when $\sigma = 10, 32, 1000$ and $1.6 \cdot 10^5$ respectively. However, when $\sigma = 1.6 \cdot 10^5$, KY consumes 424 times more memory but only 4.26% faster than ziggurat sampler in [5], where their ziggurat implementation consumes 30.57 MB memory with 2048 samples by our profiling. We use another KY implementation and test in same environment to verify their results. When $\sigma = 10^3$, it outputs 7.85 million samples/s but costs more than 200 MB memory. When $\sigma = 10^4$, the process is terminated by operating system because it costs too much memory.

The importance for developing efficient samplers for large standard deviation is that various constructions require sampling from such distributions. For constructions like homomorphic encryption, it is required to use noise-flooding technique to preserve security and privacy of circuit etc. However, various current samplers cannot deal with large standard deviation efficiently. Our efficient sampler solve this problem. This is very important for efficiency and practicality of such constructions.

We implement [9,18] fairly to test their performance. Implementation of [5] we use is what they provided in the paper. We test all implementations on same machine, compiled with same compilation flags, executes same number of times and report average performance in Table 2. Sampling speed is given in million samples per second. Time spent on generating precomputed tables is given in second.

Table 2. Performance comparison between our sampler and related works

Standard deviation	This work	Discrete zigguart ([5])		PKT ([18])		DDLL([9])
		Sampling speed	Generate CDT (s)	Sampling speed	Generate CDT (s)	
10^2	2.99	1.67	1.11	10.41	0.017	4.86
10^3	3.02	1.61	1.12	8.36	0.166	3.29
10^4	2.97	1.52	1.14	6.76	1.61	2.69
10^5	2.99	1.46	1.09	4.95	16.07	2.22
10^6	2.95	1.25	1.12	2.35	163.79	1.84
10^7	2.92	1.17	1.22	1.33	1620.8	1.64
10^8	2.93	1.04	1.11	Cost too much time		1.47

We can see that our sampler is much more efficient than [5,9,18] when standard deviation $> 10^4$. It is known that noise-flooding use much larger standard deviation than 10^4, therefore our sampler has an advantage. Moreover, our sampler and DDLL do not require additional precomputations except sampling. Before sampling operation, [5,18] first need to compute ziggurat tables and CDT respectively. This costs additional time and it is inefficient.

4 Applications: Optimizing RLWE Key Exchange and Post-quantum TLS Ciphersuite

4.1 Optimizing AKE15

Bottleneck and Our Approach. AKE15 [22] is a RLWE-variant of HMQV. It is mutual authenticated and proven secure under Bellare-Rogaway model with enhancements to capture weak perfect forward secrecy. Communicating parties do not need to encrypt or sign messages. One major bottleneck of this protocol is sampling from Gaussian distribution. According to [22], sampling operation may take up $> 60\%$ of total runtime for some parameter choices. In their implementation, generating long-term static key, polynomial c and d adopt PKT sampler. DDLL sampler is adopted in generating ephemeral keys and computing shared session key. As we discussed in Sect. 3.4, DDLL sampler is less efficient than our sampler when σ is large, thus we replace DDLL sampler with ours to sample from $D_{Z^n,\beta}$ to reduce total runtime. In one complete execution of key exchange, it requires 3 online sampling operations from $D_{Z^n,\beta}$: 2 in ephemeral key generation and 1 in shared key computation, thus our sampler can improve the efficiency of their implementation. Sampling from $D_{Z^n,\alpha}$ and $D_{Z^n,\gamma}$ still uses PKT sampler as original work. Parameter choices of the protocol remain the same and we recall them in Table 3:

Table 3. Parameter choices of AKE15 protocol

Parameter choice	Security (bits)	n	α	γ	β	Bit-length of q
I	80	1024	3.397	101.919	$8.7 \cdot 10^2$	40
II	80	2048	3.397	161.371	$4.56 \cdot 10^8$	78
III	128	2048	3.397	161.371	$1.78 \cdot 10^6$	63
IV	128	4096	3.397	256.495	$3.82 \cdot 10^{15}$	125
V	192	4096	3.397	256.495	$2.33 \cdot 10^{11}$	97
VI	256	4096	3.397	256.495	$9.12 \cdot 10^8$	81

Implementation and Performance. We report average runtime of our preliminary implementation of original AKE15 and our optimized version. Our implementation uses NTL 9.6.2, MPFR 3.1.3 and GMP 6.1.0 library with 128-bit precision. Implementation of AKE15 is executed 1,000 times and use same test environment as Sect. 3.3. Average runtime is reported in Table 4:

Table 4. Sampling and runtime of original and optimized AKE15 protocol

Parameter choice	DDLL (ms)	This work (ms)	Sampling speedup	Original AKE15 runtime (ms)	Optimized AKE15 runtime (ms)	Runtime speedup
I	0.312	0.355	0.88x	2.993	4.687	0.64x
II	1.635	0.694	2.36x	11.673	10.361	1.13x
III	1.269	0.721	1.76x	9.963	9.132	1.09x
IV	2.591	1.397	1.85x	26.741	21.964	1.22x
V	2.514	1.394	1.80x	22.865	21.457	1.07x
VI	3.349	1.394	2.40x	24.887	21.064	1.18x

By adopting our sampler, we achieve nearly 1.14x speedup of total runtime of this protocol for last 5 parameter choices. We fail to optimize parameter choice I since when σ is not large enough, our sampler is outperformed by DDLL and this leads to deceleration.

4.2 Practical Post-quantum TLS Ciphersuite

Introduction. TLS is designed to ensure secure communications over adversary controlled network, providing secrecy and data integrity between two communicating parties. It is widely deployed in real world and it already comprises more than 50% of total web traffic. It supports various algorithms for key exchange, authentication, encryption and message integrity check. Since TLS is so important and we are moving into the era of quantum computing, we consider TLS should also adopt post-quantum cryptographic primitives. However, most ciphersuites in the latest version of TLS fail to meet the demands since available key exchange and signature algorithms can be broken by quantum computers.

Our Post-quantum TLS Ciphersuite. We integrate optimized AKE15 into TLS v1.2 and this forms our post-quantum TLS ciphersuite. We give detailed cryptographic primitive combination of our ciphersuite:

- Key exchange and authentication: We integrate optimized AKE15 to achieve post-quantum key exchange and authentication. Quantum-insecure digital signatures are no longer necessary. Parameter choices follow Table 3.
- Authenticated encryption: We choose AES-128-GCM. It provides confidentiality, integrity and authenticity assurances on data.
- Hash function: We choose SHA-256. Our choice followed the principle proposed by NIST of deprecating SHA-1.

Implementation and Runtime. We use mbedTLS 1.3.10, WinNTL 9.6.2, MPFR 3.1.1 and MPIR 2.6.0 to implement our ciphersuite. Test programs simulate a TLS session between client and server. Server listens on localhost at port 443 and client communicates with local server. We measure runtime of session initiation and handshake. Test programs run in the following environment:

Lenovo ThinkCentre M8500t equipped with a 3.6 GHz Intel Core i7-4790 processor and 8 GB RAM running Windows 7 SP1 64-bit version. Test programs are compiled by Visual Studio 2010 with optimization flags and execute 1,000 times using single core. For parameter choices aimed at 80, 128, 192 and 256-bit security, average time cost is 24.417 ms, 51.224 ms, 123.443 ms and 98.842 ms respectively, communication overhead for key exchange messages is 33.125 KB, 102.25 KB, 312.25 KB and 264.5 KB respectively. In our ciphersuite, most time is spent on sending/receiving public key and key exchange messages since they are much larger than standard TLS. This might be a bottleneck of our ciphersuite.

Comparison. We compare performance of some ciphersuites in standard TLS and the post-quantum TLS ciphersuite proposed at IEEE S&P 2015 with our work. Our ciphersuite is faster in some cases but slower in others.

– Standard TLS v1.2: We choose two standard TLS ciphersuites: 0x9F (1024-bit DH+2048-bit RSA) and 0xC030 (elliptic curve secp521r1+2048-bit RSA). Test environment and procedure remain the same as Sect. 4.2. Runtime of these two ciphersuites are 30.959 ms and 49.742 ms respectively. For comparison, our 80-bit parameter choice I is faster than ciphersuite 0x9F, 256-bit parameter choice VI is slower than ciphersuite 0xC030.
– BCNS15: This work introduced implementation of an unauthenticated post-quantum key exchange aimed at 128-bit security and integration in TLS protocol. We implement client/server side test programs using code in [8] and test these ciphersuites: RLWE-RSA-AES128-GCM-SHA256 and RLWE-ECDSA-AES128-GCM-SHA256. Test environment remain the same as Sect. 3.3. For first ciphersuite, server adopts a self-signed 3072-bit RSA certificate and average execution time is 44.536 ms. For the second ciphersuite, server adopts a self-signed ECDSA certificate using curve secp256k1 and average execution time is 41.539 ms. Our post-quantum TLS ciphersuite at same 128-bit security is slower and average runtime is 51.224 ms. Our ciphersuite has much larger communication overhead than this work (around 10 KB). Another difference is that our ciphersuite can achieve mutual authentication while this work only authenticates the server. Furthermore, we use different library and operating system to test, thus it is harder to compare directly and fairly. We believe their ciphersuites have better performance and smaller communication cost, but ours is more closer to a fully post-quantum TLS ciphersuite.

5 Conclusion

In this paper, we introduce a much faster discretized Gaussian sampler based on the ziggurat sampling algorithm. We utilize several optimization techniques to improve our sampler, so that our sampler has advantage on computation efficiency. We prove that the statistical distance between distribution generated by our sampler and discrete Gaussian distribution is very small so that our sampler is suitable for lattice-based cryptography. We also present optimized implementation and comparisons with several related samplers. Results show that our sampler is very computational efficient, especially when σ is large. This can benefit constructions using noise-flooding technique (e.g., homomorphic encryption). We also give two applications: first is optimizing RLWE-based authenticated key exchange protocol. We achieve 1.14x speedup on total runtime of this protocol over major parameter choices. Another application is we present our practical post-quantum TLS ciphersuite. Performance of ciphersuite is close to standard TLS v1.2 ciphersuites and BCNS15. We believe our sampler and post-quantum TLS ciphersuite will have further optimizations and more applications.

Acknowledgement. We would like to thank Jiang Zhang for valuable help and discussions, Chen Feng for the support on this paper. We also thank anonymous reviewers for valuable feedbacks. Implementation for testing Knuth-Yao sampler is from Rachid El Bansarkhani. This work is supported by National Natural Science Foundation of China (Grant No. 61402035) and Fundamental Research Funds for the Central Universities (Grant No. 2014JBM033, No. 2015YJS039 and No. 2017YJS038).

References

1. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
2. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (1997)
3. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7
4. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: IEEE Symposium on Security and Privacy (SP), 2015, pp. 553–570. IEEE (2015)
5. Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., Weiden, P.: Discrete ziggurat: a time-memory trade-off for sampling from a Gaussian distribution over the integers. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 402–417. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_20
6. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
7. Ding, J., Xie, X., Lin, X.: A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology EPrint Archive, 2012:688 (2012)
8. Douglas, S.: dstebila/openssl-rlwekex
9. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
10. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. Appl. Algebra Eng. Commun. Comput. **25**(3), 159–180 (2014)

11. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
12. Knuth, D.: The complexity of nonuniform random number generation. In: Algorithms and Complexity, New Directions and Results, pp. 357–428 (1976)
13. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_33
14. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
15. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
16. Marsaglia, G., Tsang, W.W., et al.: The ziggurat method for generating random variables. J. Stat. Softw. **5**(8), 1–7 (2000)
17. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: Proceedings of the forty-first annual ACM symposium on Theory of computing, pp. 333–342. ACM (2009)
18. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_5
19. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (JACM) **56**(6), 34 (2009)
20. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, 1994 Proceedings, pp. 124–134. IEEE (1994)
21. Weiden, P., Hülsing, A., Cabarcas, D., Buchmann, J.A.: Instantiating treeless signature schemes. IACR Cryptology ePrint Archive, 2013:65 (2013)
22. Zhang, J., Zhang, Z., Ding, J., Snook, M., Dagdelen, Ö.: Authenticated key exchange from ideal lattices. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 719–751. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_24