Designs, Codes and Cryptography (2023) 91:1779–1811
https://doi.org/10.1007/s10623-022-01164-7

Polar sampler: A novel Bernoulli sampler using polar codes with application to integer Gaussian sampling

Jiabo Wang[1] · **Cong Ling**[2]

Received: 2 January 2022 / Revised: 17 July 2022 / Accepted: 6 December 2022 /
Published online: 13 January 2023

Abstract
Cryptographic constructions based on hard lattice problems have emerged as a front runner for the standardization of post-quantum public-key cryptography. As the standardization process takes place, optimizing specific parts of proposed schemes, e.g., Bernoulli sampling and integer Gaussian sampling, becomes a worthwhile endeavor. In this work, we propose a novel Bernoulli sampler based on polar codes, dubbed "polar sampler". The polar sampler is information theoretically optimum in the sense that the number of uniformly random bits it consumes approaches the entropy bound asymptotically. It also features quasi-linear complexity and constant-time implementation. An integer Gaussian sampler is developed using multilevel polar samplers. Our algorithm becomes effective when sufficiently many samples are required at each query to the sampler. Security analysis is given based on Kullback–Leibler divergence and Rényi divergence. Experimental and asymptotic comparisons between our integer Gaussian sampler and state-of-the-art samplers verify its efficiency in terms of entropy consumption, running time and memory cost. We envisage that the proposed Bernoulli sampler can find other applications in cryptography in addition to Gaussian sampling.

Keywords Bernoulli sampling · Discrete Gaussian sampling · Polar codes · Integer lattice · Kullback–Leibler divergence · Rényi divergence · Constant-time

Mathematics Subject Classification 94A20 · 94A60 · 68P30

1 Introduction

Lattice-based cryptography is one of the most promising candidates of cryptosystems in the plausible post-quantum age. The security of lattice-basedprimitives is guaranteed by the

Communicated by K. Matsuura.

✉ Cong Ling
c.ling@imperial.ac.uk

Jiabo Wang
wangjiabo.2013@tsinghua.org.cn

[1] Strategic Centre for Research in Privacy-Preserving Technologies and Systems, Nanyang Technological University, Singapore 637553, Singapore

[2] Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, UK

hardness of worst-case lattice problems, e.g. the Learning With Errors (LWE) problem [22, 34] and Short Integer Solution (SIS) problem [24, 25]. The discrete Gaussian distribution lies at the core of security proofs of these primitives, and it is also one of the fundamental building blocks of practical lattice-based cryptographic applications, e.g. signature schemes, encryption and key exchanges. In general, the security level of these cryptographic applications is closely related to the statistical performance of the discrete Gaussian sampling (DGS) algorithm. From an implementation standpoint, cryptographers also take other qualities of a DGS into consideration including side-channel resistance, computation and storage efficiency. In practice, the trade-off between these performances is a bottleneck of this problem.

It has been widely assumed that for cryptographic applications with λ bits of security the statistical distance (SD) between the ideal distribution and the approximated one should be roughly $2^{-\lambda}$ such that there is only minor loss in security [12]. Some other measures such as Kullback–Leibler (KL) divergence and Rényi divergence are proved to provide more efficient security analysis than SD, as they can lower the requirement for precision and reduce the cost of the algorithms in many practical cases [5, 30–32]. From a practical point of view, the difficulty of DGS lies in the implementation of DGS in cryptographic primitives with constrained resources. Besides the resilience against potential side-channel attacks, designers looking for the optimal DGS solution to a specific application must strike the balance of memory consumption and running time, precision and efficiency.

There are already a variety of works addressing the application of DGS in lattice-based primitives. Existing techniques include the binary sampling [11], the cumulative distribution table (CDT) sampler [10], the Knuth–Yao sampler [20], and the discrete Ziggurat sampler [23], etc. In [14], rejection sampling is used to generate discrete Gaussian samples where one draws an element x from a discrete domain uniformly at random and accepts it with probability proportional to $\exp(-x^2/2\sigma^2)$ where σ is the standard deviation. However, calculating the exponential function requires high-precision computing and sufficient trials are needed before the sampler produces an output. In [11], a CDT is used as a base sampler and the rejection sampling is done in a bitwise manner to produce discrete Gaussian samples for BLISS. However, the CDT sampling itself takes 35 percent of the total running time of BLISS [19] and the precomputed table for rejection sampling requires larger memory when a wider distribution is in need.

In [18], Hülsing et al. replaced the discrete Gaussian distribution by a rounded Gaussian distribution in Lyubashevsky's signature scheme without trapdoors and BLISS showing its effectiveness, security and efficiency. As the term suggested, a rounded Gaussian distribution is derived by rounding continuous Gaussian samples which can be efficiently realized by Box–Muller transform [7] in constant time. A convolution method, first proposed in [28], can expand a discrete Gaussian distribution with a small parameter to a wider one. Another sampling design [26] exploits a base sampler with small parameters to efficiently generate DGS with arbitrary and varying parameters in a convolutional manner. This application-independent algorithm consists of an online and offline stage, both of which can be carried out in constant time, proving a resilience against timing attack. A constant-time sampler was proposed in [39] and Rényi divergence was used to improve the efficiency. In [35], arithmetic coding, a classical data compression technique, was adopted as a sampler in BLISS giving a reduced signature size.

When reviewing the literature of DGS, we find that Bernoulli sampling is of vital importance to randomness generation. It is involved in many cryptographic designs and a typical example is BLISS [11] where Bernoulli sampling is employed to build a discrete Gaussian sampler. To make BLISS safe under side channel attacks, especially the timing-based one, improved Bernoulli samplers were devised in [8, 13, 29, 39]. To improve the efficiency of

Bernoulli sampling with biases in exponential or cosh form, as is the case in BLISS, polynomial approximation with sufficient precision were proposed in [6, 39]. Our research begins with Bernoulli sampling and we get our inspiration from polar source coding. The proposed Bernoulli sampler can be used for generating arbitrary discrete distribution and this paper is concerned about its application to Gaussian sampling.

1.1 Contribution

In this work, we propose a novel Bernoulli sampler using polar codes and apply it to DGS over the integers. Polar codes are the first class of efficiently encodable and decodable codes which provably achieve channel capacity of symmetric channels [3]. It can also achieve Shannon's data compression rate [4]. The power of polar codes stems from the polarization phenomenon: under Arıkan's polar transform, information measures of synthesized sources (or channels) converge to either 0 or 1 when coding becomes trivial. Moreover, the state-of-the-art decoding runs with $O(N \log \log N)$ complexity where N denotes the block length of a polar code [38]. Given their attractive performance, polar codes have found a wide range of applications in information theory and communication systems. In particular, they have been standardized for the fifth-generation (5G) wireless communication networks.

This work tackles the sampling problem from a source coding perspective, namely, sampling can be considered the inverse problem of source coding. In source coding or data compression, one typically encodes a block of symbols of a certain distribution into some bits which become uniformly random as the block length tends to infinity [9]. Since a source code is invertible, inverting this process would produce samples from the desired distribution. Polar sampling is well suited for applications where a large number of independent discrete Gaussian samples are required (e.g. fully homomorphic encryption (FHE), digital signatures) as the averaged randomness consumption per sample decreases asymptotically to the optimum thanks to the polarization effect. Note that the polar sampler is not restricted to sampling from the discrete Gaussian distribution, but can be extended to other distributions of interest in cryptography.

The principal contributions of this paper are summarized as follows:

- A novel approach to sample from a Bernoulli distribution as well as an integer Gaussian sampler using multilevel polar samplers are developed. Using a binary partition tree, we recursively partition $\mathbb{Z}$ into 2 cosets, 4 cosets, and so on. The number of partitions is only logarithmic in s. Each partition gives rise to a binary source, which is produced by one polar sampler. The advantage of this multilevel sampling approach is that only Bernoulli samples are needed, which allows simpler implementation than sampling over the whole integer domain.
- Analysis of approximation errors. Although multilevel polar samplers would produce the desired distribution $D_{\mathbb{Z}^N,c,s}$, it is not exactly so. This is because the polar sampler converts N i.i.d. Bernoullis into N polarized and unpolarized Bernoullis. We approximate the polarized ones using either unbiased or deterministic Bernoullis which will only yield an approximate version of the desired distribution. We derive upper bounds on the closeness between the target discrete Gaussian and its approximation measured by KL divergence.
- Security analysis. To achieve a certain security level in a standard cryptographic scheme with oracle access to a discrete Gaussian distribution, the principle of setting the parameters of our polar sampler is also discussed based on KL divergence. In cryptographic applications where the number of queries q to the Gaussian sampler is limited (e.g., $q \leq 2^{64}$ in the NIST specifications of signatures), using Rényi divergence can yield con-

siderable savings according to previous work of [5, 32]. We also apply Rényi divergence to improve the parameter selection of polar sampler.

The proposed multilevel polar sampler scheme complements and distinguishes from existing discrete Gaussian samplers in the literature. In addition to offering a different approach, it exhibits several salient features:

- Information theoretic optimality. Asymptotically, the multilevel polar sampler achieves the entropy bound of the discrete Gaussian distribution. This implies that it requires minimum resources of random bits to produce the desired distribution.
- Quasi-linear complexity. The proposed Gaussian sampling approach enjoys low complexity. The design of a polar sampler can be done at the offline stage, that is, given a target distribution, it is done once and for all. The online stage of a polar sampler computes certain posterior probabilities which can be implemented in $O(N \log N)$ complexity.[1] We also give experimental and asymptotic comparison between our DGS approach and other existing samplers including Knuth–Yao sampling, binary sampling and CDT sampling. The prominent advantage of multilevel polar sampler is the entropy consumption which indicates the cost of randomness. The overall running time depends on both SC decoding(computing LRs) and Bernoulli sampling. Compared with the binary sampling [11], polar sampler has higher computational complexity but it asymptotically and effectively reduces the entropy consumption. We also illustrate in experiments that the multilevel polar sampler is faster than Knuth–Yao sampling.
- Constant-time implementation. The proposed discrete Gaussian sampler is constant-time in the sense that the running time is independent of the output values. This makes our sampler attractive when dealing with timing side-channel attacks. From a perspective of coding theory, polar sampler is constant-time because polar codes have a fixed code length which compares favorably with other source coding techniques (e.g. Huffman coding).

Of course, the proposed sampler can be combined with existing "expander" techniques such as convolution if needed. In this work, we focus on the theoretic design and analysis of polar samplers, whereas various optimization issues (e.g., concrete computational/storage costs etc.) are left to future work. Nevertheless, we have found it in experiments that even a prototype implementation significantly outperforms the Knuth–Yao sampler in speed in benchmark experiments.

1.2 Roadmap

The roadmap of this paper is given as follows. Section 2 introduces the proposed Bernoulli sampler, i.e., polar sampler, and elucidates its relation to polar source coding. Section 3 presents how we devise an integer Gaussian sampler using multiple polar samplers. Section 4 analyses the approximation error of our integer Gaussian sampler based on KL divergence. In Sect. 5, the security, precision and parameter selection are discussed based on KL and Rényi divergence. Section 6 gives a comprehensive analysis of the constant-time feature and compares our integer Gaussian sampler with state-of-the-art samplers regarding the complexity. Section 7 concludes this paper.

[1] It can be upgraded to $O(N \log \log N)$ using the state-of-the-art SC decoding [38].

2 Bernoulli sampling using polar codes

2.1 Notation

Given a vector $x^{1:N}$ and a set $\mathcal{A} \subset \{1, \ldots, N\}$, we denote by $x_{\mathcal{A}}$ the subvector of $x^{1:N}$ indexed by $\mathcal{A}$ and denote by $x^{(i)}$ the i-th coordinate of $x^{1:N}$. A capital letter is used to denote a variable while its lowercase represents a realization. Denote by $X \sim P$ a distribution P of X over a countable set $\mathcal{X}$. Then the entropy of X is defined as $H_P(X) = -\sum_{x\in\mathcal{X}} p(x) \log p(x)$. We write $H(X) = H_P(X)$ for brevity if the distribution is clear. Suppose X and Y have a joint distribution $P(X, Y)$. The conditional entropy of X given Y is defined as $H(X|Y) = \sum_{x\in\mathcal{X}, y\in\mathcal{Y}} p(x, y) \log \frac{p(y)}{p(x,y)}$. The logarithm to base 2 is denoted by log while the natural logarithm is denoted by ln.

2.2 Source polarization

The key idea of polar source coding can be found in [4] where a polar code was proposed to achieve Shannon's source coding bound. Let $(X^{1:N}, Y^{1:N})$ denotes N i.i.d. copies of a memoryless source (X, Y) of joint distribution $P_{X,Y}$, where X takes values over $\mathcal{X} = \{0, 1\}$ while Y takes values over a countable set $\mathcal{Y}$. The two random source X and Y are correlated, and Y is called the side-information. In source coding, the encoder compresses a sequence $X^{1:N}$ into a shorter codeword such that the decoder can yield an estimation $\hat{X}^{1:N}$ of $X^{1:N}$ given the shorter codeword and side information $Y^{1:N}$.[2]

Polar codes are proved to achieve Shannon's source coding bound asymptotically. The source polarization transform from $X^{1:N}$ to $U^{1:N}$ is performed by applying an entropy-preserving circuit to $X^{1:N}$, i.e.,

$$U^{1:N} = X^{1:N} G_N,\ G_N = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n} B_N,$$

where $^{\otimes n}$ denotes the n-th Kronecker power, and B_N is a bit-reversal permutation [3] of the input vector. Figure 1 illustrates the source polarization transform of $X^{1:2}$ and $X^{1:4}$ where $\oplus$ denotes mod-2 sum. This transform preserves the entropy in the sense that

$$H(U^{1:2} \mid Y^{1:2}) = 2H(X \mid Y),\ H(U^{1:4} \mid Y^{1:4}) = 4H(X \mid Y).$$

Meanwhile, it also polarizes the entropy in the sense that

$$H(U^{(1)} \mid Y^{1:4}) \geq H(S^{(1)} \mid Y^{1:2}) = H(S^{(2)} \mid Y^{3:4}) \geq H(U^{(2)} \mid Y^{1:4}, U^{(1)}),$$

and

$$\begin{aligned} H(U^{(3)} \mid Y^{1:4}, U^{1:2}) \geq & H(R^{(1)} \mid Y^{1:2}, S^{(1)}) \\ = & H(R^{(2)} \mid Y^{3:4}, S^{(2)}) \geq H(U^{(4)} \mid Y^{1:4}, U^{1:3}). \end{aligned}$$

By applying the construction in Fig. 1 recursively, we derive a bijection $U^{1:N} = X^{1:N} G_N$ inducing a combined source pair $(U^{1:N}, Y^{1:N})$ and a transition $W_N : U^{1:N} \to Y^{1:N}$. This combined source pair is then split into N synthesized source pairs $(U^{(i)}, Y^{1:N} \times U^{1:i-1})$ giving rise to N sub-transitions $W_N^{(i)} : U^{(i)} \to Y^{1:N} \times X^{1:i-1}$. A polarization phenomenon happened to sub-source pairs is observed and stated as follows.

[2] Polar source coding still holds in the absence of side information.

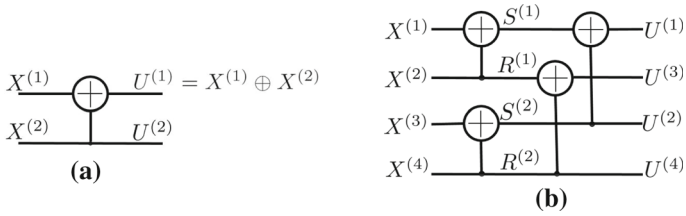

Fig. 1 The source polarization transform [3]: **a** A two-by-two transform. **b** A four-by-four transform

Theorem 1 (Source polarization [4]) *Let* (X, Y) *be a source as above. For any* $N = 2^n, n \geq 1$, *let* $U^{1:N} = X^{1:N} G_N$. *Then, for any* $0 < \beta < 0.5$, *as* $N \to \infty$,

$$\frac{\left|\left\{i \in [1, N] : H(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in (1 - 2^{-N^\beta}, 1]\right\}\right|}{N} \to H(X \mid Y), \tag{1}$$

$$\frac{\left|\left\{i \in [1, N] : H(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in [0, 2^{-N^\beta})\right\}\right|}{N} \to 1 - H(X \mid Y). \tag{2}$$

Note that in the absence of side information Y, the above theorem still holds by considering Y independent of X.

Definition 1 (*Bhattacharyya parameter* [15]) Let $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ be a pair of random variables where $\mathcal{X} = \{0, 1\} = \mathrm{GF}(2)$ and $\mathcal{Y}$ is an arbitrary finite set. Let X and Y follow the joint distribution $P_{XY}(x, y)$. If X is the source to be compressed and Y is the side information, the Bhattacharyya parameter is defined as

$$\begin{aligned} Z(X|Y) &\equiv 2 \sum_y P_Y(y) \sqrt{P_{X|Y}(0|y) P_{X|Y}(1|y)} \\ &= 2 \sum_y \sqrt{P_{X,Y}(0, y) P_{X,Y}(1, y)}. \end{aligned} \tag{3}$$

Proposition 1 ([4], Proposition 2)

$$(Z(X|Y))^2 \leq H(X|Y), \tag{4}$$

$$H(X|Y) \leq \log(1 + Z(X|Y)). \tag{5}$$

It is implied by Proposition 1 that for a source (X, Y), the parameters $H(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ and $Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ polarize simultaneously in the sense that $H(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ approaches 0 (resp. 1) as $Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ approaches 0 (resp. 1).

For $\beta \in (0, 1/2)$ and $\alpha = 2^{-N^\beta}$, the indexes of $U^{1:N}$ can be divided into a low-entropy set

$$\mathcal{L}_{X|Y} = \left\{i \in [N] : \ Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in [0, \alpha)\right\}, \tag{6}$$

and its complement $\mathcal{L}^c{}_{X|Y}$. Again, in the absence of side information Y, the two sets are defined in the same way by considering Y independent of X and U.

This gives rise to the encoding and decoding scheme described in [4]. More specifically, for a realization of $(X^{1:N}, Y^{1:N}) = (x^{1:N}, y^{1:N})$, the encoder computes $u^{1:N} = x^{1:N} G_N$ and

only shares $u_{\mathcal{L}^c_{X|Y}}$ with the decoder. The compression rate is defined as $R = |\mathcal{L}^c_{X|Y}|/N$. The decoder can obtain an estimate $\hat{u}^{1:N}$ of $u^{1:N}$ in a successive manner as

$$\hat{u}^{(i)} = \begin{cases} u^{(i)}, & \text{if } i \in \mathcal{L}^c_{X|Y} \\ 0, & \text{if } i \in \mathcal{L}_{X|Y} \text{ and } L_N^{(i)}(y^{1:N}, \hat{u}^{1:i-1}) \geq 1 \\ 1, & \text{if } i \in \mathcal{L}_{X|Y} \text{ and } L_N^{(i)}(y^{1:N}, \hat{u}^{1:i-1}) < 1 \end{cases}, \tag{7}$$

where $L_N^{(i)}(y^{1:N}, \hat{u}^{1:i-1})$ is called the likelihood ratio (LR) defined by

$$L_N^{(i)}(y^{1:N}, \hat{u}^{1:i-1}) = \frac{P(U^{(i)} = 0 | Y^{1:N} = y^{1:N}, U^{1:i-1} = \hat{u}^{1:i-1})}{P(U^{(i)} = 1 | Y^{1:N} = y^{1:N}, U^{1:i-1} = \hat{u}^{1:i-1})}. \tag{8}$$

Theorem 2 (An upper bound on error probability [4]) *For any fixed $R > H(X|Y)$ and $\beta < 0.5$, the probability of error for the above polar source coding method is bounded as $P_e = \Pr(\hat{U}^{1:N} \neq U^{1:N}) = O(2^{-N^\beta})$.*

It implies that any compression rate $R > H(X|Y)$ is achievable with a vanishing block error probability for polar source coding of sufficiently large N. As N goes to infinity, the polarization process removes the randomness of the low-entropy set almost surely while the other set becomes random. Additionally, the complexity of polar encoding and decoding are both $O(N \log N)$.

2.3 From source coding to Bernoulli sampling

Given the notion of memoryless source $(X, Y) \sim P_{X,Y}$ and polar source coding, we now consider the sampling problem, i.e., to sample from a Bernoulli distribution $P(X)$ given (or without) side information Y. We propose a novel Bernoulli sampler called PolarSampler$(\cdot)$ in Algorithm 1. The interfaces, key operations and subroutines are described as follows.

2.3.1 Interfaces

PolarSampler$(\cdot)$ draws N samples $x^{1:N}$ from the target distribution $P(X)$ given N samples $y^{1:N}$ of the side information $Y \in \mathcal{Y} := \{1, 2, \ldots, |\mathcal{Y}|\}$. It has access to a precomputed table $\mathbf{c} = [c_1, \ldots, c_{|\mathcal{Y}|}]$ where each element indicates a Bernoulli bias $c_y = P(X = 1 | Y = y)$ and y is the index of c_y in $\mathbf{c}$. The bias vector $\mathbf{c}[y^{1:N}]$ is defined as $[c_{y^{(1)}}, \ldots, c_{y^{(N)}}]$ whose elements take values in $\mathbf{c}$ indexed by side information vector $y^{1:N} = [y^{(1)}, \ldots, y^{(N)}]$.

In addition to the low-entropy set $\mathcal{L}_{X|Y}$ as defined in formula (6), we further define a high-entropy set.[3]

$$\mathcal{H}_{X|Y} = \left\{ i \in [N] : \ Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in (1 - \alpha, 1] \right\}, \tag{9}$$

where $\alpha = 2^{-N^\beta}$. In order to define the two sets $\mathcal{H}_{X|Y}$ and $\mathcal{L}_{X|Y}$, one needs to calculate the Bhattacharyya parameter $Z(U^{(i)}|Y^{1:N}, U^{1:i-1})$ efficiently. However, as a source pair (X, Y) turns into a synthesized source pair $(U^{(i)}, Y^{1:N} \times U^{1:i-1})$ by polarization transform, the alphabet size of the side information increases exponentially with N. Calculating

[3] In [4], $\mathcal{L}^c{}_{X|Y}$ is called the high-entropy set which is larger than $\mathcal{H}_{X|Y}$ in (9); more details will be given in Fig. 2.

input : $N = 2^n$,$y^{1:N}$,$\mathcal{H}_{X|Y}$, $\mathcal{L}_{X|Y}$ (Or $N = 2^n$, $1^{1:N}$, $\mathcal{H}_X$, $\mathcal{L}_X$ without Y.)
output: $x^{1:N}$

Define global arrays:LRReg $[N][n+1]$, UReg $[N][n+1]$;
LRReg [:][0]=$(1-\mathbf{c}[y^{1:N}])/\mathbf{c}[y^{1:N}]$; // Or LRReg [:][0]=$(1-\mathbf{c}[1^{1:N}])/\mathbf{c}[1^{1:N}]$ without Y
for $i \leftarrow 1$ **to** N **do**
  LRReg $\leftarrow$ CalLR(n,i);
  if *index* $i \in \mathcal{H}_{X|Y}$ **then**
    UReg$[i][n] \leftarrow$randomBin(); // formula (10).
  end
  else if *index* $i \in \mathcal{L}_{X|Y}$ **then**
    UReg$[i][n] =$ LRReg$[i][n] < 1$; // formula (10).
  end
  else index $i \in \mathcal{H}^c_{X|Y} \backslash \mathcal{L}_{X|Y}$
    UReg$[i][n] =$Uniform() $< 1/(1+$ LRReg$[i][n])$; // Uniform() produces values in (0,1] uniformly at random; formula (11).
  end
  UReg $\leftarrow$ CalBit(n,i); // UReg[:][0] $= U^{1:N}G_N$
end
return $x^{1:N} =$ UReg[:][0]

Algorithm 1: PolarSampler(·).

$Z(U^{(i)}|Y^{1:N}, U^{1:i-1})$ according to how we define it in Definition 1 becomes intractable. Some efficient algorithms to calculate the Bhattacharyya parameters were proposed in [36].[4] In addition, preparing the bias table $\mathbf{c}$ and calculating $Z(U^{(i)}|Y^{1:N}, U^{1:i-1})$ are done offline. We will refer to the offline stage as the construction stage of PolarSampler(·) in the sequel.

Note that when there is no side information Y, the precomputed table $\mathbf{c}$ only consists of one bias $c_1 = P(X = 1)$. The bias vector will be $\mathbf{c}[1^{1:N}] = [c_1, \cdots, c_1]$ of length N. The high- and low-entropy sets $\mathcal{H}_X$ and $\mathcal{L}_X$ are defined in the same manner by considering Y independent of X.

2.3.2 Key operations

According to polar source coding, the $U^{(i)}$ for $i \in \mathcal{H}_{X|Y}$ with very high entropy is approximately uniformly distributed and is approximately independent of both $U^{1:i-1}$ and $Y^{1:N}$, while the $U^{(i)}$ for $i \in \mathcal{L}_{X|Y}$ with very low entropy is almost deterministic. Those $U^{(i)}$ for $i \in \mathcal{H}^c_{X|Y} \backslash \mathcal{L}_{X|Y}$ (i.e., $Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in [\alpha, 1-\alpha]$) are unpolarized. As N goes to infinity, the fraction of unpolarized indexes vanishes and the fraction of high-entropy indexes approaches the entropy $H(X|Y)$ of X given Y according to Theorem 1.

Since the polarization transform G_N is invertible with $G_N^{-1} = G_N$, it is expected to produce an approximation $Q_{X^{1:N}}$ of $P_{X^{1:N}}$ by applying the above transform to $U^{1:N}$, i.e. $X^{1:N} = U^{1:N}G_N$. However, the unpolarized set may not be negligible for finite length N, and should be handled with care. More precisely, those $U^{(i)}$ for $i \in \mathcal{H}^c_{X|Y} \backslash \mathcal{L}_{X|Y}$ are neither uniform nor deterministic; assigning values inaccurately will cause non-negligible distortion of the target distribution. To minimize the distortion, $U^{(i)}$ should obey the following

[4] Matlab codes available in the folder .../PolarFastSCL/HowToConstructPolarCode at https://github.com/YuYongRun/PolarCodeDecodersInMatlab.git.

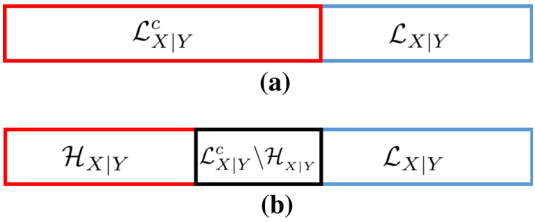

Fig. 2 Polar source coding vs. polar sampling: **a** Subsets of indexes for polar source coding. **b** Subsets of indexes for polar sampling. The fraction of $\mathcal{L}^c_{X|Y}\backslash\mathcal{H}_{X|Y}$ vanishes as N goes to infinity

distribution:

$$U^{(i)} = \begin{cases} \{0,1\} \sim \text{Bernoulli}(0.5), & \text{if } i \in \mathcal{H}_{X|Y} \\ \arg\max_u P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(u|y^{1:N},u^{1:i-1}), & \text{if } i \in \mathcal{L}_{X|Y} \end{cases}, \tag{10}$$

and

$$U^{(i)} = \begin{cases} 0 \;\; w.p. \; P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(0|y^{1:N},u^{1:i-1}) \\ 1 \;\; w.p. \; P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(1|y^{1:N},u^{1:i-1}) \end{cases} \text{if } i \in \mathcal{H}^c_{X|Y}\backslash\mathcal{L}_{X|Y} \quad . \tag{11}$$

Figure 2 shows the difference between source coding and sampling: although the unpolarized set belongs to the compressed codeword in source coding, its bits should be randomized as in (11) in sampling.

While sampling according to formula (10) is obviously trivial and straightforward, the bottleneck of entropy consumption is determined by sampling the unpolarized set in formula (11). The following lemma is adapted from [27, Lemma 1] which gives the fraction of unpolarized set for finite $n = \log N$.

Lemma 1 (Fraction of unpolarized set [27]) *Let μ ($3.579 \le \mu \le 4.714$) be a constant called the upper bound on the scaling exponent which is solely determined by the conditional probability $P_{X|Y}$. For a constant $v > 1$ and $n = \log N \ge 1$, the following relation holds:*

$$\frac{\left|\left\{i \in [1,N] : Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in [2^{-vn},\; 1-2^{-vn}]\right\}\right|}{N} < c2^{-n/\mu},$$

where the constant c depends solely on v and it does not depend on n or $P_{X|Y}$.

Corollary 1 (Asymptotic property of polarization) *Let $X \sim P(X)$ be the target Bernoulli distribution with side information Y. As $N \to \infty$, the fraction of $\mathcal{H}_{X|Y}$ goes to $H(X|Y)$, the fraction of $\mathcal{L}_{X|Y}$ goes to $1 - H(X|Y)$ and the fraction of $\mathcal{L}^c_{X|Y}\backslash\mathcal{H}_{X|Y}$ scales as $N^{-\frac{1}{\mu}}$ for* $3.579 \le \mu \le 4.714$.

In the absence of side information Y, the above property also holds by substituting X for $X|Y$.

Proof Recall it from Proposition 1 that $Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ and $H(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ polarize simultaneously. Therefore Theorem 1 can be restated as follows. For any $0 < \beta < 0.5$, as $N \to \infty$,

$$\frac{\left|\mathcal{H}_{X|Y} := \left\{i \in [1,N] : Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in (1-2^{-N^\beta}, 1]\right\}\right|}{N} \to H(X \mid Y)$$

$$\frac{\left|\mathcal{L}_{X|Y} := \left\{i \in [1,N] : Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1}) \in [0, 2^{-N^\beta})\right\}\right|}{N} \to 1 - H(X \mid Y).$$

Given a threshold 2^{-N^β}, we can find a v such that $N^{-v} = 2^{-N^\beta}$. Lemma 1 implies that the Bhattacharyya parameter $Z(U^{(i)} \mid Y^{1:N}, U^{1:i-1})$ falls on the interval $[N^{-v}, 1 - N^{-v}]$ with a probability smaller than $cN^{-1/\mu}$ where $3.579 \leq \mu \leq 4.714$ and c is determined by v and $P(X|Y)$. Therefore, the fraction of unpolarized set $\mathcal{L}^c_{X|Y} \backslash \mathcal{H}_{X|Y}$ scales as $N^{-1/\mu}$. The above conclusions still hold in the absence of side information Y by considering X independent of Y. Q.E.D. □

2.3.3 Subroutines

To carry out the operations in formula (10) and (11), one also needs to calculate $P_{U^{(i)}|Y^{1:N},U^{1:i-1}}$. Recall that the definition of likelihood ratio is

$$L_N^{(i)}(y_1^N, u^{1:i-1}) = \frac{P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(0|y^{1:N}, u^{1:i-1})}{P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(1|y^{1:N}, u^{1:i-1})}. \tag{12}$$

Since these likelihood ratios can be computed with quasi-linear complexity $O(N \log N)$ by SC decoding proposed in [3], the posterior probability $P_{U^{(i)}|Y^{1:N},U^{1:i-1}}$ in (10) and (11) can be equivalently computed. When there is no side information Y, we can compute $P_{U^{(i)}|U^{1:i-1}}$ in the same manner as above by considering Y independent of X.

In PolarSampler(·), we first define a two-dimensional likelihood ratio array LRReg[N][$n+1$] and a two-dimensional bit array UReg[N][$n+1$] indexed by integers $1 \leq i \leq N$ and $0 \leq m \leq n$. We also define an array of $N \times (n+1)$ nodes connected by multiple 2-by-2 butterfly circuits "⧖" as in Fig. 3 for $N = 8$. Each node takes the responsibility to update a unique element of LRReg[N][$n+1$] and a unique element of UReg[N][$n+1$] of the same index. We define two properties of each layer of the array, i.e., phase and branch denoted by integers ϕ and ψ, respectively. In Fig. 3, we distinguish different phases at each layer by different colors. At layer m, the phase and branch satisfy $1 \leq \phi \leq 2^m$ and $0 \leq \psi < 2^{n-m}$. Note that for any layer m, each index $1 \leq i \leq 2^n$ has a unique representation as

$$i = \langle \phi, \psi \rangle_m = \phi + 2^m \cdot \psi.$$

And for a generic array A we abbreviate $A[\langle \phi, \psi \rangle_m][m]$ as $A[\langle \phi, \psi \rangle][m]$. To ease the notations, we also use the notation of likelihood ratio $L_{2^m}^{(\phi)}$ to denote the node by which it is calculated. In Line 2 of PolarSampler(·), the raw likelihood ratios $L_1^{(1)} = P(X = 0|y)/P(X = 1|y)$ are stored in LRReg[:][0] given side information samples $y^{1:N}$. Other elements of LRReg[i][m] = LRReg[$\langle \phi, \psi \rangle$][$m$] and UReg[$i$][$m$] = UReg[$\langle \phi, \psi \rangle$][$m$] for $m > 1$ will be uniquely calculated by node $L_{2^m}^{(\phi)}$ of phase ϕ and branch ψ at layer m. After PolarSampler(·) finishes its work, the likelihood ratios in formula (12) are finally stored in the n-th column LRReg[:][n] of LRReg and the bit vector $U^{1:N}$ is finally stored in UReg[:][n].

The subroutines CalLR(·) and CalBit(·) are employed to recursively calculate and update the likelihood ratio array LRReg[N][$n+1$] and the bit array UReg[N][n+1]. This process is exactly what SC decoding does as proposed in [3, Sect. VIII] and modularized in [37, Sect. II].

As a high level description, CalLR(·) recursively assembles two likelihood ratios of the same phase but different branches at layer $m-1$ (two nodes on RHS of "⧖") and derive two new likelihood ratios of different phases but the same branch at layer m (two nodes on LHS

input : m, ϕ
output: updated LRReg

1 **if** $m = 0$ **then return** set $\kappa = \lceil \phi/2 \rceil$;
2 **if** $\phi \mod 2 = 1$ **then** `CalLR` $(m-1, \kappa)$ **for** $\psi = 0, \cdots, 2^{n-m} - 1$ **do**
3 **if** $\phi \mod 2 = 1$ **then** $\mathsf{LRReg}[\langle \phi, \psi \rangle][m] \xleftarrow{\text{equation(13)}}$ $(\mathsf{LRReg}[\langle \kappa, 2\psi \rangle][m-1], \mathsf{LRReg}[\langle \kappa, 2\psi+1 \rangle][m-1])$ **else** temp=$\mathsf{UReg}[\langle \phi - 1, \psi \rangle][m]$; $\mathsf{LRReg}[\langle \phi, \psi \rangle][m] \xleftarrow{\text{equation(14)}} (\mathsf{LRReg}[\langle \kappa, 2\psi \rangle][m-1], \mathsf{LRReg}[\langle \kappa, 2\psi+1 \rangle][m-1])$
4 **end**

Algorithm 2: The $\mathsf{CalLR}(m, \phi)$ function.

input : m, ϕ
output: updated UReg

1 **if** $\phi \mod 2 = 1$ **then return** set $\kappa = \phi/2$;
2 **for** $\psi = 0, \cdots, 2^{n-m} - 1$ **do**
3 $\mathsf{UReg}[\langle \kappa, 2\psi \rangle][m-1] \leftarrow \mathsf{UReg}[\langle \phi - 1, \psi \rangle][m] \oplus \mathsf{UReg}[\langle \phi, \psi \rangle][m]$;
4 $\mathsf{UReg}[\langle \kappa, 2\psi+1 \rangle][m-1] \leftarrow \mathsf{UReg}[\langle \phi, \psi \rangle][m]$;
5 **end**
6 **if** $\kappa \mod 2 = 0$ **then** `CalBit` $(m-1, \kappa)$

Algorithm 3: The $\mathsf{CalBit}(m, \phi)$ function.

of "⧖") according to formula (13) and (14) as[5]

$$\mathrm{LRReg}[\langle 2\kappa - 1, \psi \rangle][m] = \frac{\mathrm{LRReg}[\langle \kappa, 2\psi \rangle][m-1] \cdot \mathrm{LRReg}[\langle \kappa, 2\psi+1 \rangle][m-1] + 1}{\mathrm{LRReg}[\langle \kappa, 2\psi \rangle][m-1] + \mathrm{LRReg}[\langle \kappa, 2\psi+1 \rangle][m-1]}, \tag{13}$$

$$\mathrm{LRReg}[\langle 2\kappa, \psi \rangle][m] = \left[\mathrm{LRReg}[\langle \kappa, 2\psi \rangle][m-1]\right]^{1-2\cdot\mathrm{temp}} \cdot \mathrm{LRReg}[\langle \kappa, 2\psi+1 \rangle][m-1], \tag{14}$$

where $\mathrm{temp} = \mathrm{UReg}[\langle 2\kappa - 1, \psi \rangle][m]$. CalBit$(\cdot)$ works in the other way around as in Line 4 and 5 of Algorithm 3 which gives two bits $\mathrm{UReg}[\langle \kappa, 2\psi \rangle][m-1]$ and $\mathrm{UReg}[\langle \kappa, 2\psi+1 \rangle][m-1]$ of the same phase but different branches at layer $m-1$ (two nodes on RHS of "⧖") given two bits $\mathrm{UReg}[\langle 2\kappa - 1, \psi \rangle][m]$ and $\mathrm{UReg}[\langle 2\kappa, \psi \rangle][m]$ of different phases but the same branch at layer m (two nodes on LHS of "⧖").

We give an example to demonstrate how CalLR$(\cdot)$ and CalBit$(\cdot)$ work in Fig. 3. In PolarSampler$(\cdot)$, CalLR(n, i) begins with $i = 1$ for node $L_8^{(1)}$. It in turn activates two $L_4^{(1)}$ nodes at layer 2, then four $L_2^{(1)}$ nodes at layer 1 and terminates at eight $L_1^{(1)}$ nodes at layer 0. The nodes at layer 0 pass their likelihood ratios $\mathrm{LRReg}[\langle 1, \psi \rangle][0]$ for $0 \le \psi \le 7$ to the blue nodes at layer 1 where new likelihood ratios $\mathrm{LRReg}[\langle 1, \psi \rangle][1]$ for $\psi = 0, 1, 2, 3$ are computed according to formula (13). Likewise, the newly computed likelihood ratios are passed forward and computed until node $L_8^{(1)}$ are finally reached and $\mathrm{LRReg}[\langle 1, 0 \rangle][3]$ is updated. All the nodes activated so far are blue nodes in Fig. 3. At iteration $i = 2$, node $L_8^{(2)}$ does not activate any node but computes $\mathrm{LRReg}[\langle 2, 0 \rangle][3]$ according to formula (14) given the values of $\mathrm{LRReg}[\langle 1, 0 \rangle][2]$ and $\mathrm{LRReg}[\langle 1, 1 \rangle][2]$ already calculated by two $L_4^{(1)}$ nodes as well as the value of $\mathrm{UReg}[\langle 1, 0 \rangle][3]$. Then node $L_8^{(2)}$ updates $\mathrm{UReg}[\langle 2, 0 \rangle][3]$ as in line 6

[5] One may concern about the floating-point divisions for safety reason. The LR recursions can be equivalently replaced by division-free probability recursions of the same complexity; see Appendix B for further details.

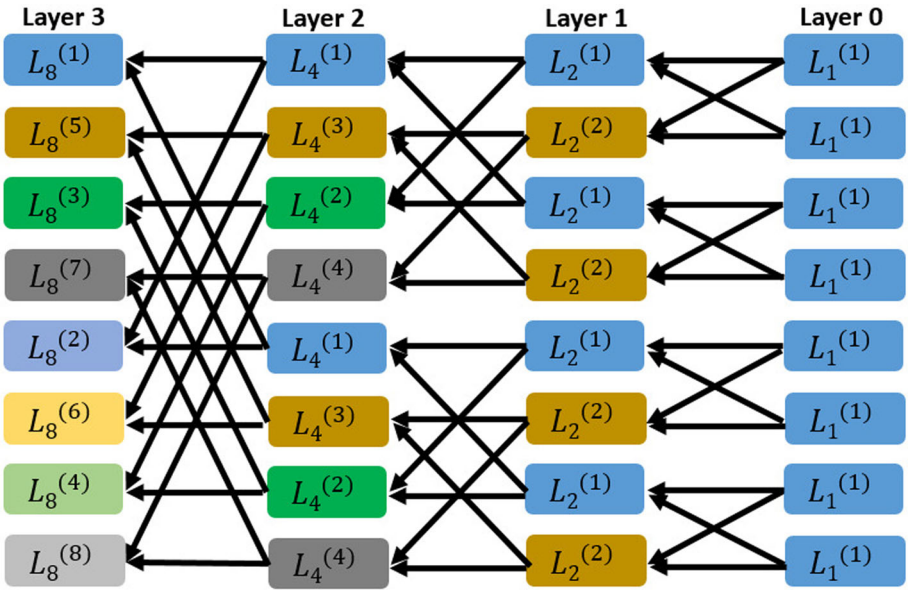

Fig. 3 The butterfly circuit

or 9 or 12 of Algorithm 1. Given UReg[$\langle 1, 0\rangle$][3] and UReg[$\langle 2, 0\rangle$][3], CalBit($\cdot$) calculates UReg[$\langle 1, 0\rangle$][2] and UReg[$\langle 1, 1\rangle$][2] using the 2-by-2 transform $G_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. CalBit($\cdot$) is suspended for $i = 2$. It cannot proceed to activate the four $L_2^{(1)}$ nodes to update the bit array UReg because UReg[$\langle 2, 0\rangle$][2] and UReg[$\langle 2, 1\rangle$][2] are yet to be available.

For every iteration i, CalLR($\cdot$) activates all the nodes in the same phase (i.e., the same color) and updates the corresponding elements in LRReg[N][$m + 1$]. CalBit($\cdot$) recursively calculates UReg[$\langle \kappa, 2\psi\rangle$][$m - 1$], UReg[$\langle \kappa, 2\psi + 1\rangle$][$m - 1$] if both UReg[$\langle 2\kappa - 1, \psi\rangle$][$m$] and UReg[$\langle 2\kappa, \psi\rangle$][$m$] are available. Every node in the array is activated once to update LRReg and one more time to update UReg leading to an overall computational complexity $O(N \log N)$.

2.4 Closeness analysis of polar sampler

A good closeness metric can help reduce the complexity of implementation. In this section, we will evaluate the closeness error of the proposed sampling scheme.

Definition 2 *(Kullback–Leibler divergence)* Let P and Q be two distributions over a common countable set Ω, and let $\mathcal{A} \subset \Omega$ be the strict support of P ($P(a) > 0$ iff. $a \in \mathcal{A}$). The KL divergence D_{KL} of Q from P is defined as:

$$D_{KL}(P \| Q) = \sum_{a \in \mathcal{A}} P(a) \ln \left(\frac{P(a)}{Q(a)} \right),$$

with the convention that $\ln(x/0) = +\infty$ for any $x > 0$.

Let $P_{X^{1:N}}(x^{1:N})$ denote the distribution of N i.i.d. Bernoullis X defined as above. For any $0 < \beta < 0.5$, $N = 2^n$, $n \geq 1$ and the corresponding high- and low-entropy sets defined in

(9) and (6), one can generate a distribution $Q_{X^{1:N}}(x^{1:N})$ using the rules (10) and (11). To give the KL divergence between $P_{X^{1:N}}(x^{1:N})$ and $Q_{X^{1:N}}(x^{1:N})$, we first modify the Bernoulli sampling rules (10) and (11) to be

$$U^{(i)} = \begin{cases} 0 \ w.p. \ \frac{1}{2} \\ 1 \ w.p. \ \frac{1}{2} \end{cases} \quad \text{if } i \in \mathcal{H}_{X|Y} \ , \tag{15}$$

$$U^{(i)} = \begin{cases} 0 \ w.p. \ P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(0|y^{1:N}, u^{1:i-1}) \\ 1 \ w.p. \ P_{U^{(i)}|Y^{1:N},U^{1:i-1}}(1|y^{1:N}, u^{1:i-1}) \end{cases} \quad \text{if } i \in \mathcal{H}^c_{X|Y} \ , \tag{16}$$

where only the deterministic decisions for $U^{(i)}$ in $\mathcal{L}_{X|Y}$ are replaced by random decisions. Let $Q'_{X^{1:N}}(x^{1:N})$ denote the distribution derived by the new Bernoulli sampling rules described in (15) and (16).

Theorem 3 (Polar sampling theorem) *Let $X \sim P(X)$ be the target Bernoulli distribution with bias $c_1 = P(X = 1)$. To sample from the Bernoulli distribution $Ber(c_1)$, PolarSampler$(\cdot)$ in Algorithm 1* takes a bias table $\mathbf{c} = [c_1, \cdots, c_1]$ together with the high- and low-entropy set $\mathcal{H}_X, \mathcal{L}_X$ as input and computes N new Bernoulli biases $b_1, b_2, \ldots, b_N$ for a 2-power N. Sampling independent variables $U^{1:N}$ according to biases $b_1, b_2, \ldots, b_N$ and applying the transform G_N lead to a vector $X^{1:N} = U^{1:N}G_N$ of distribution $Q_{X^{1:N}}$. Let $Q'_{X^{1:N}}$ be the intermediate distribution defined earlier in this section. Then for $0 < \beta < 0.5$, we derive

$$D_{KL}(P_{X^{1:N}} \| Q'_{X^{1:N}}) \leq 2 \ln 2 \cdot N 2^{-N^\beta} \ and \quad D_{KL}(Q_{X^{1:N}} \| Q'_{X^{1:N}}) \leq \ln 2 \cdot N 2^{-N^\beta}.$$

In the presence of a side information $Y \in \mathcal{Y} := \{y_1, \cdots, y_{|\mathcal{Y}|}\}$, X can be seen as a combination of a sequence of Bernoullis with a bias table $\mathbf{c} = [c_1, \ldots, c_{|\mathcal{Y}|}]$ for $c_y = P(X = 1|Y = y)$. PolarSampler$(\cdot)$ takes the side information $y^{1:N}$ and the high- and low-entropy set $\mathcal{H}_{X|Y}, \mathcal{L}_{X|Y}$ as input. The above closeness bound still holds by substituting $X|Y$ for X.

Proof The KL divergence between $P_{X^{1:N}}$ and $Q'_{X^{1:N}}$ is bounded by the KL divergence between $P_{U^{1:N}|Y^{1:N}}$ and $Q'_{U^{1:N}|Y^{1:N}}$ because the following relation hold.

$$\begin{aligned} D_{KL}\left(P_{X^{1:N}} \| Q'_{X^{1:N}}\right) &\leq D_{KL}\left(P_{X^{1:N},Y^{1:N}} \| Q'_{X^{1:N},Y^{1:N}}\right) \\ &\overset{(a)}{=} D_{KL}\left(P_{Y^{1:N}} \| Q'_{Y^{1:N}}\right) + D_{KL}\left(P_{X^{1:N}|Y^{1:N}} \| Q'_{X^{1:N}|Y^{1:N}}\right) \\ &\overset{(b)}{=} D_{KL}\left(P_{X^{1:N}|Y^{1:N}} \| Q'_{X^{1:N}|Y^{1:N}}\right) \overset{(c)}{=} D_{KL}\left(P_{U^{1:N}|Y^{1:N}} \| Q'_{U^{1:N}|Y^{1:N}}\right). \end{aligned} \tag{17}$$

The above equalities are derived as follows.
(a) The chain rule of KL divergence.
(b) $D_{KL}(P_{Y^{1:N}} \| Q'_{Y^{1:N}}) = 0$.
(c) One-to-one mapping between $U^{1:N}$ and $X^{1:N}$.

The conditional KL divergence $D_{KL}\left(P_{U^{1:N}|Y^{1:N}} \| Q'_{U^{1:N}|Y^{1:N}}\right)$ is derived as

$$\begin{aligned} &D_{KL}\left(P_{U^{1:N}|Y^{1:N}} \| Q'_{U^{1:N}|Y^{1:N}}\right) \\ &\overset{(d)}{=} \sum_{i=1}^{N} D_{KL}\left(P_{U^{(i)}|U^{1:i-1},Y^{1:N}} \| Q'_{U^{(i)}|U^{1:i-1},Y^{1:N}}\right) \end{aligned}$$

$$\stackrel{(e)}{=} \sum_{i\in\mathcal{H}_k} D_{KL}\left(P_{U^{(i)}|U^{1:i-1},Y^{1:N}} \| Q'_{U^{(i)}|U^{1:i-1},Y^{1:N}}\right)$$

$$\stackrel{(f)}{=} \sum_{i\in\mathcal{H}_k} \ln 2\left[1 - H_P\left(U^{(i)}|U^{1:i-1}, Y^{1:N}\right)\right]$$

$$\stackrel{(g)}{\leq} \sum_{i\in\mathcal{H}_k} \ln 2\left[1 - Z_P\left(U^{(i)}|U^{1:i-1}, Y^{1:N}\right)^2\right] \tag{18}$$

$$\stackrel{(h)}{\leq} 2\ln 2 \cdot N2^{-N^\beta}, \tag{19}$$

where the equalities and inequalities are explained as follows.
(d) The chain rule of KL divergence.
(e) For $i \in \mathcal{H}^c_{X|Y}$, $Q'(u^{(i)}|u^{1:i-1}, y^{1:N}) = P(u^{(i)}|u^{1:i-1}, y^{1:N})$.
(f) The definition of $D_{KL}(\cdot\|\cdot)$ and $Q'(U^{(i)}|u^{1:i-1}, y^{1:N}) = \frac{1}{2}$ for $i \in \mathcal{H}_{X|Y}$.
(g) $Z(X|Y)^2 \leq H(X|Y)$ [4].
(h) Definition of $\mathcal{H}_{X|Y}$ (9).
In a similar fashion, the KL divergence of $Q_{X_{1:r}^{1:N}}$ and $Q'_{X_{1:r}^{1:N}}$ is as follows.

$$D_{KL}\left(Q_{X^{1:N}} \| Q'_{X^{1:N}}\right) \leq D_{KL}\left(Q_{U^{1:N}|Y^{1:N}} \| Q'_{U^{1:N}|Y^{1:N}}\right)$$

$$= \sum_{i=1}^{N} D_{KL}\left(Q_{U^{(i)}|U^{1:i-1},Y^{1:N}} \| Q'_{U^{(i)}|U^{1:i-1},Y^{1:N}}\right)$$

$$\stackrel{(i)}{=} \sum_{i\in\mathcal{L}_{X|Y}} D_{KL}\left(Q_{U^{(i)}|U^{1:i-1},Y^{1:N}} \| Q'_{U^{(i)}|U^{1:i-1},Y^{1:N}}\right)$$

$$\stackrel{(j)}{=} \sum_{i\in\mathcal{L}_{X|Y}} \ln 2 \sum_{u^{1:i-1},y^{1:N}} -Q'\left(u^{1:i-1}, y^{1:N}\right) \log Q'\left(\bar{u}^{(i)}|U^{1:i-1}, Y^{1:N}\right)$$

$$\stackrel{(k)}{\leq} \sum_{i\in\mathcal{L}_{X|Y}} \ln 2 \cdot H_P\left(U^{(i)}|U^{1:i-1}, Y^{1:N}\right)$$

$$\stackrel{(l)}{\leq} \sum_{i\in\mathcal{L}_{X|Y}} \ln 2 \cdot Z\left(U^{(i)}|U^{1:i-1}, Y^{1:N}\right) \tag{20}$$

$$\stackrel{(m)}{\leq} \ln 2 \cdot N2^{-N^\beta}, \tag{21}$$

where the equalities and inequalities come from
(i) For $i \in \mathcal{L}^c_{X|Y}$, $Q'\left(u^{(i)}|u^{1:i-1}, y^{1:N}\right) = Q\left(u^{(i)}|u^{1:i-1}, y^{1:N}\right)$.
(j) The definition of $D_{KL}(\cdot\|\cdot)$ (see Appendix A).
(k) See Appendix A.
(l) $H(X|Y) \leq Z(X|Y)$ [15].
(m) Corollary 1.

Therefore, the closeness measured by KL divergence can be concluded as

$$D_{KL}(P_{X^{1:N}} \| Q'_{X^{1:N}}) \leq 2\ln 2 \cdot N2^{-N^\beta} \text{ and } D_{KL}(Q_{X^{1:N}} \| Q'_{X^{1:N}}) \leq \ln 2 \cdot N2^{-N^\beta}.$$

Note that polar sampling without side information is easier. In the absence of Y, the above closeness analysis for KL divergence still hold by seeing Y independent of X.

Q.E.D. □

Although we cannot give the KL divergence between $P_{X^{1:N}}$ and $Q_{X^{1:N}}$ due to the lack of triangle inequality, the absence of $D_{KL}(P_{X^{1:N}} \| Q_{X^{1:N}})$ will not prevent us from giving the security analysis which will be explained in Sect. 5.

3 Gaussian sampling over the integers using polar sampler

Definition 3 For any $c \in \mathbb{R}$, $s > 0$, define the discrete Gaussian distribution over $\mathbb{Z}$ as

$$\forall x \in \mathbb{Z},\ D_{\mathbb{Z},c,s}(x) = \rho_{c,s}(x)/\rho_{c,s}(\mathbb{Z}),$$

where $\rho_{c,s}(x) = \exp(-\pi|x-c|^2/s^2)$ and $\rho_{c,s}(\mathbb{Z}) = \sum_{z\in\mathbb{Z}} \rho_{c,s}(z)$.

In the above definition, the denominator $\rho_{c,s}(\mathbb{Z})$ is for normalization. For convenience, we may omit c for $c = 0$, e.g. $\rho_{0,s}(x) = \rho_s(x)$ and $D_{\mathbb{Z},0,s}(x) = D_{\mathbb{Z},s}(x)$.

Gaussian sampling over the integers $\mathbb{Z}$ can be formulated as a multilevel sampling problem over a binary partition chain $\mathbb{Z} \subset 2\mathbb{Z} \subset 4\mathbb{Z} \subset \cdots 2^r\mathbb{Z} \cdots$ of which each level is labeled by $X_1, X_2, \ldots, X_r, \ldots$ (see Fig. 4). Then the discrete Gaussian distribution over the integers $\mathbb{Z}$ induces a distribution $P_{X_{1:r}}$ whose limit is exactly $D_{\mathbb{Z},c,s}$ as r goes to infinity. By cutting off the tail area of negligible probability, a discrete Gaussian distribution over the integer lattice $\mathbb{Z}$ can be reduced to a distribution over a finite set. For example, if the cutoff points of $D_{\mathbb{Z},0,s=3\sqrt{2\pi}}$ are ± 16, the left and right tail areas are approximately 2^{-20}.

input : $\mathbf{c} = [\mathbf{c}_1, \cdots, \mathbf{c}_r], \mathcal{H}_{X_k|X_{1:k-1}}, \mathcal{L}_{X_k|X_{1:k-1}}$ for $k = 1, \cdots, r$.
output: $x^{1:N}$

1 temp = $1^{1:N}$;
2 $x_1^{1:N}$=PolarSampler(temp,$\mathcal{H}_{X_1}$,$\mathcal{L}_{X_1}$); // It has access to the precomputed table $\mathbf{c}_1 = P(X_1 = 1)$.
3 temp = $x_1^{1:N}$;
4 $x_2^{1:N}$=PolarSampler(temp,$\mathcal{H}_{X_2|X_1}$,$\mathcal{L}_{X_2|X_1}$); // It has access to $\mathbf{c}_2 = [P(X_2 = 1|X_1 = 0), P(X_2 = 1|X_1 = 1)]$.
5 $\cdots$
6 temp = $x_{1:r-1}^{1:N}$;
7 $x_r^{1:N}$=PolarSampler(temp,$\mathcal{H}_{X_r|X_{1:r-1}}$,$\mathcal{L}_{X_r|X_{1:r-1}}$);// It has access to $\mathbf{c}_r = [P(X_r = 1|X_{r-1}, \cdots, X_1 = 0\cdots 0), \cdots, P(X_r = 1|X_{r-1}, \cdots, X_1 = 1\cdots 1)]$
8 **return** $x^{1:N} = x_1^{1:N} + 2 \cdot x_2^{1:N} + \cdots + 2^{r-1} \cdot x_r^{1:N}$

Algorithm 4: GaussianSampling(·).

We now begin to demonstrate how PolarSampler(·) can be used for discrete Gaussian sampling as in Algorithm 4 GaussianSampling(·). Suppose r levels of partition are employed to approximate $D_{\mathbb{Z},c,s}$. The chain rule of conditional probability and the chain rule of conditional entropy, i.e.

$$P(X_{1:r}) = \prod_{k=1}^{r} P(X_k|X_{1:k-1}) \text{ and } H(X_{1:r}) = \sum_{k=1}^{r} H(X_k|X_{1:k-1}),$$

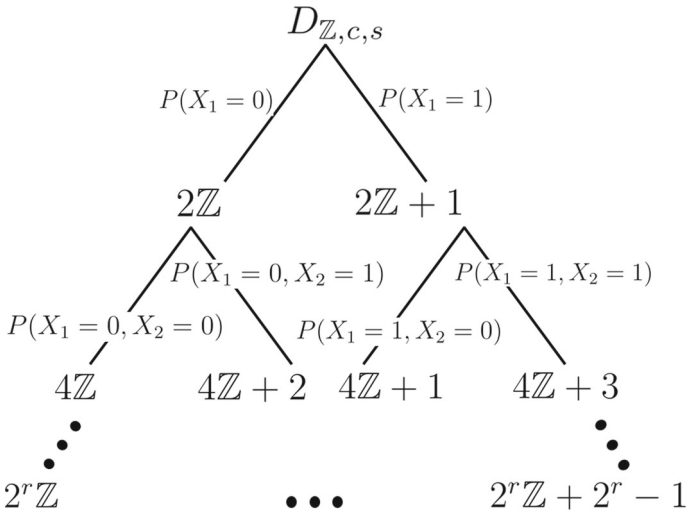

Fig. 4 An r-level binary partition tree of the integer lattice $\mathbb{Z}$

imply that the Gaussian distribution over the finite constellation can be generated in a level-by-level way. For the k-th level, we can sample from the component source X_k using PolarSampler$(\cdot)$ given the samples $x_{1:k-1}$ from lower levels as side information. GaussianSampling$(\cdot)$ has access to a bias table $\mathbf{c} = [\mathbf{c}_1, \cdots, \mathbf{c}_r]$ defined as follows.

$$\begin{aligned}
&\mathbf{c}_1 = [P(X_1 = 1)]_{1\times 1}\\
&\mathbf{c}_2 = [P(X_2 = 1|X_1 = 0),\, P(X_2 = 1|X_1 = 1)]_{2\times 1}\\
&\quad\cdots\\
&\mathbf{c}_r = [P(X_r = 1|X_{1:r-1} = 0\cdots 0), \ldots, P(X_r = 1|X_{1:r-1} = 1\cdots 1)]_{2^{r-1}\times 1}.
\end{aligned}$$

The high- and low-entropy sets $\mathcal{H}_{X_k|X_{1:k-1}}$ and $\mathcal{L}_{X_k|X_{1:k-1}}$ are computed according to the bias table $\mathbf{c}$ offline. We call this stage the construction stage of GaussianSampling$(\cdot)$. The online stage of GaussianSampling$(\cdot)$ draws Bernoulli samples level by level using PolarSampler$(\cdot)$ and we call it the implementation stage.

For the first level, we want to generate the component source X_1 in the absence of any side information.

1. *Construction* By performing the source polarization transformation G_N on N i.i.d. copies of X_1, we obtain an N dimensional vector $U_1^{1:N} = X_1^{1:N} G_N$. For any $\beta \in (0, 1/2)$ and $\alpha = 2^{-N^\beta}$, we formally define two sets $\mathcal{H}_{X_1}$ and $\mathcal{L}_{X_1}$ as

$$\mathcal{H}_{X_1} = \left\{ i \in [N] :\ Z(U_1^{(i)} \mid U_1^{1:i-1}) \in (1-\alpha, 1] \right\}, \tag{22}$$

and

$$\mathcal{L}_{X_1} = \left\{ i \in [N] :\ Z(U_1^{(i)} \mid U_1^{1:i-1}) \in [0, \alpha) \right\}. \tag{23}$$

For any $i \in \mathcal{H}_{X_1}$, $U_1^{(i)}$ is approximately uniform and independent of $U_1^{1:i-1}$, while for $i \in \mathcal{L}_{X_1}$, $U_1^{(i)}$ is almost deterministic given the knowledge of $U_1^{1:i-1}$.

2. *Implementation* As discussed in Sect. 2.3, PolarSampler(·) calculates the posterior probability $P_{U_1^{(i)}|U_1^{1:i-1}}$ using SC decoding. Given the two sets $\mathcal{H}_{X_1}$, $\mathcal{L}_{X_1}$ and $P_{U_1^{(i)}|U_1^{1:i-1}}$, PolarSampler(·) generates N i.i.d. samples of X_1 by applying the polarization transform circuit to the vector $U_1^{1:N}$ of which each entry takes a value according to the following rule:

$$U_1^{(i)} = \begin{cases} \text{Bernoulli}(\frac{1}{2}) & \text{if } i \in \mathcal{H}_{X_1} \\ \arg\max_{u_1^{(i)}} P_{U_1^{(i)}|U_1^{1:i-1}}(u_1^{(i)}|u_1^{1:i-1}) & \text{if } i \in \mathcal{L}_{X_1} \end{cases}, \tag{24}$$

and

$$U_1^{(i)} = \begin{cases} 0 \ w.p. \ P_{U_1^{(i)}|U_1^{1:i-1}}(0|u_1^{1:i-1}) \\ 1 \ w.p. \ P_{U_1^{(i)}|U_1^{1:i-1}}(1|u_1^{1:i-1}) \end{cases} \text{if } i \in \mathcal{H}_{X_1}^c \backslash \mathcal{L}_{X_1} . \tag{25}$$

Once we have a realization $u_1^{1:N}$ of $U_1^{1:N}$, we derive a realization $x_1^{1:N} = u_1^{1:N} G_N$ of $X_1^{1:N}$ and pass it to the next level for further processing.

For higher levels with $k \in (1, r]$, our task is to generate N i.i.d. samples of source X_k given the side information $x_{1:k-1}^{1:N}$ which were generated at the previous $k-1$ levels.

1. *Construction* By performing the source polarization transformation circuit G_N on N i.i.d. copies of X_k, we obtain an N dimensional vector $U_k^{1:N} = X_k^{1:N} G_N$. For $\beta \in (0, 1/2)$ and $\alpha = 2^{-N^\beta}$, we define $\mathcal{H}_{X_k|X_{1:k-1}}$ and $\mathcal{L}_{X_k|X_{1:k-1}}$ as

$$\mathcal{H}_{X_k|X_{1:k-1}} = \left\{ i \in [N] : \ Z(U_k^{(i)} \mid X_{1:k-1}^{1:N}, U_k^{1:i-1}) \in (1-\alpha, 1] \right\}, \tag{26}$$

$$\mathcal{L}_{X_k|X_{1:k-1}} = \left\{ i \in [N] : \ Z(U_k^{(i)} \mid X_{1:k-1}^{1:N}, U_k^{1:i-1}) \in [0, \alpha) \right\}. \tag{27}$$

2. *Implementation* Again PolarSampler(·) calculates $P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}$ using SC decoding. Then it generates N i.i.d. copies of X_k by applying the polarization transformation circuit to vector $U_k^{1:N}$ of which each entry takes a value according to the following rule:

$$U_k^{(i)} = \begin{cases} \text{Bernoulli}\left(\frac{1}{2}\right) & \text{if } i \in \mathcal{H}_{X_k|X_{1:k-1}} \\ \arg\max_u P_{U^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}\left(u|x_{1:k-1}^{1:N}, u_k^{1:i-1}\right) & \text{if } i \in \mathcal{L}_{X_k|X_{1:k-1}} \end{cases}, \tag{28}$$

$$U_k^{(i)} = \begin{cases} 0 \ w.p. \ P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}\left(0|x_{1:k-1}^{1:N}, u_k^{1:i-1}\right) \\ 1 \ w.p. \ P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}\left(1|x_{1:k-1}^{1:N}, u_k^{1:i-1}\right) \end{cases},$$

$$\text{if } i \in \mathcal{H}_{X_k|X_{1:k-1}}^c \backslash \mathcal{L}_{X_k|X_{1:k-1}}. \tag{29}$$

Once we have a realization $u_k^{1:N}$ of $U_k^{1:N}$, we derive a realization $x_k^{1:N} = u_k^{1:N} G_N$ of $X_k^{1:N}$ and pass it to the next level for further processing. Recall that the approximation error for each level is determined by parameter β and N. To achieve a target closeness between the ideal distribution and the one we can produce, the two sets $\mathcal{H}_{X_k|X_{1:k-1}}$ and $\mathcal{L}_{X_k|X_{1:k-1}}$ for each level are properly chosen and determined offline. By repeating the operations in (28) and (29) from level 2 to level r, we can finally obtain N samples $x^{1:N}$ from $D_{\mathbb{Z},c,s}$, i.e.,

$$x^{1:N} = \sum_{k=1}^{r} 2^{k-1} x_k^{1:N}. \tag{30}$$

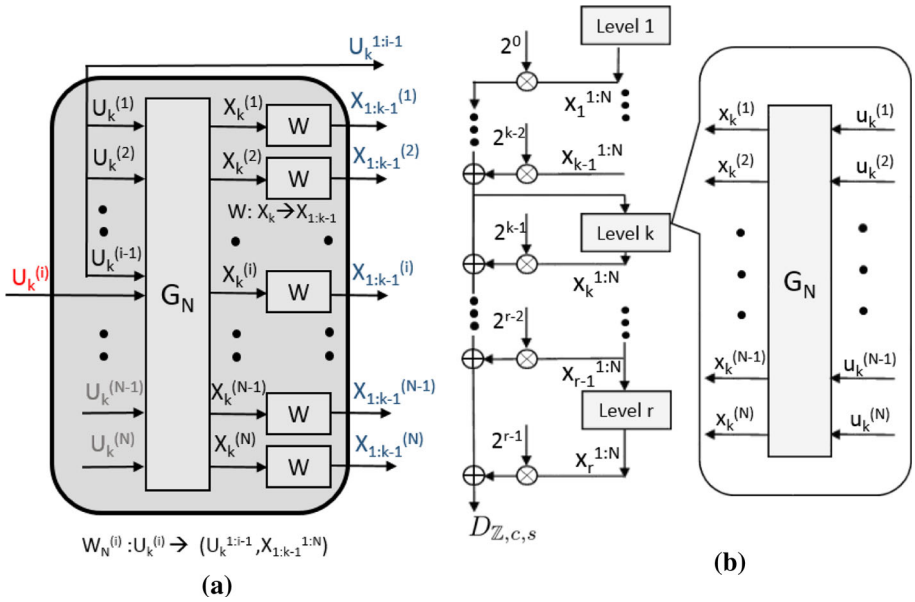

Fig. 5 The construction and implementation of the GaussianSampling(·): **a** Construction (runs offline). **b** Implementation (runs online)

Figure 5 shows how this Gaussian sampler works at each level in terms of construction and implementation. It also shows how to combine the output of each level. At the construction stage, W designates the probability transition from X_k to $X_{1:k-1}$ and $W_N^{(i)}$ designates the transition of synthesized source pair $(U_k^{(i)}, U_k^{1:i-1} \times X_{1:k-1}^{1:N})$. At this stage the Bhattacharyya parameters of $W_N^{(i)}$ are calculated to define $\mathcal{H}_{X_k|X_{1:k-1}}$ and $\mathcal{L}_{X_k|X_{1:k-1}}$. At the implementation stage, realizations of $U^{1:N}$ are produced according to the implementation rules (28) and (29). Given the two parameters N and β, the closeness between the ideal distribution and the one our sampler can produce will be analysed in next section.

4 Closeness analysis of discrete Gaussian sampling

4.1 The approximation error model

In a concrete implementation, an ideal discrete Gaussian distribution is replaced by an approximation. To give a sharp estimation of the accuracy/security of a cryptographic primitive, the closeness between the ideal distribution and its approximation should be measured. In this section, we will derive the upper bounds on the closeness between the ideal distribution and the one generated by our sampling scheme measured by KL divergence.

The approximation error comes from two sources, the tailcut (owing to finite levels of partitions) and the polar sampling. On the one hand, we need to decide how many levels of partitions are needed. On the other hand, the error introduced by polar sampling should also be analysed. Denote by $D_{\mathbb{Z},c,s}$ the target discrete Gaussian distribution and we decide to employ r levels of partition. If polar sampling did not introduce any error, we would generate a distribution $P_{X_{1:r}}$ with a closeness measure $\delta(D_{\mathbb{Z},c,s}, P_{X_{1:r}})$ which is determined

only by r for some metric δ. In reality, polar sampling produces a distribution $Q_{X_{1:r}}(x_{1:r})$ and it introduces an error of $\delta(P_{X_{1:r}}, Q_{X_{1:r}})$. We will in this section analyse the above two closeness quantities using Kullback–Leibler (KL) divergence.

Since the KL divergence does not satisfy the triangle inequality, we will give $D_{KL}(D_{\mathbb{Z},c,s} \| P_{X_{1:r}})$ and $D_{KL}(P_{X_{1:r}} \| Q_{X_{1:r}})$ separately rather than a total KL divergence $D_{KL}(D_{\mathbb{Z},c,s} \| Q_{X_{1:r}})$. However, as discussed in [31, Chapter 3] the lack of symmetry and triangle inequality can be handled by a KL-based security argument which will be presented in Sect. 5.

4.2 Approximation error from tailcut

Definition 4 (*Smoothing parameter* [25]) For an n-dimensional lattice Λ, and positive real $\epsilon > 0$, we define its smoothing parameter $\eta_\epsilon(\Lambda)$ to be the smallest s such that $\rho_{1/s}(\Lambda^* \backslash \{0\}) \leq \epsilon$.

The smoothing parameter quantifies how large s must be for $D_{\Lambda,c,s}$ to behave like a continuous Gaussian distribution. It is implied by Definition 4 that for any $\epsilon > 0$, the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ of $\mathbb{Z}$ is the smallest s such that $\rho(s\mathbb{Z}) \leq 1 + \epsilon$.

Lemma 2 ([14, Lemma 4.2]) *For any $\epsilon > 0$, any $s > \eta_\epsilon(\mathbb{Z})$, and any $t > 0$,*

$$\Pr_{x \leftarrow D_{\mathbb{Z},c,s}}(|x - c| \geq t \cdot s) \leq \frac{1+\epsilon}{1-\epsilon} \cdot 2e^{-\pi t^2}.$$

Instead of sampling over the full domain of the integer lattice, a distribution tail of negligible probability is cut off in practice. Suppose 2^r samples are left after the tailcut. Let $\mathcal{A} = \mathbb{Z} \cap [-2^{r-1} + c, 2^{r-1} + c)$. The distribution of the finite set is

$$D_\gamma(a) = \frac{\rho_{c,s}(a)}{\sum_{a \in \mathcal{A}} \rho_{c,s}(a)} = D_{\mathbb{Z},c,s}(a) / D_{\mathbb{Z},c,s}(\mathcal{A}),$$

where $a \in \mathcal{A}$ and γ is the probability of the tail. This constellation $\mathcal{A}$ of 2^r points can be represented as a binary partition tree labeled by $X_{1:r}$ in the same way as Fig. 4. In our sampling scheme, we derive a sample labeled by

$$x^{1:N} = \sum_{k=1}^{r} 2^{k-1} x_k^{1:N}.$$

There exists a one-to-one mapping from $X_{1:r}$ to $\mathcal{A}$. Therefore $P_{X_{1:r}}$ and the tailcut distribution D_γ are exactly the same and we can obtain $D_{KL}(D_{\mathbb{Z},c,s} \| P_{X_{1:r}})$ by calculating $D_{KL}(D_{\mathbb{Z},c,s} \| D_\gamma)$. The distribution D_γ over the finite constellation $\mathcal{A}$ can be written in the form

$$P(X_{1:r} = x) = D_{\mathbb{Z},c,s}(a) / \sum_{a \in \mathcal{A}} D_{\mathbb{Z},c,s}(a) = D_{\mathbb{Z},c,s}(a | a \in \mathcal{A}).$$

The KL divergence between D_γ and $D_{\mathbb{Z},c,s}$ is

$$\begin{aligned} D_{KL}(D_\gamma \| D_{\mathbb{Z},c,s}) &= \sum_{a \in \mathcal{A}} D_{\mathbb{Z},c,s}(a | a \in \mathcal{A}) \ln \frac{D_{\mathbb{Z},c,s}(a | a \in \mathcal{A})}{D_{\mathbb{Z},c,s}(a)} \\ &= \sum_{a \in \mathcal{A}} D_{\mathbb{Z},c,s}(a | a \in \mathcal{A}) \ln \frac{D_{\mathbb{Z},c,s}(a | a \in \mathcal{A})}{D_{\mathbb{Z},c,s}(a | a \in \mathcal{A}) D_{\mathbb{Z},c,s}(x \in \mathcal{A})} \end{aligned}$$

$$= \sum_{a \in \mathcal{A}} D_{\mathbb{Z},c,s}(a|a \in \mathcal{A}) \ln \frac{1}{D_{\mathbb{Z},c,s}(a \in \mathcal{A})} = \ln \frac{1}{D_{\mathbb{Z},c,s}(a \in \mathcal{A})}.$$

According to the second-order Taylor bound, if $D_{\mathbb{Z},c,s}(a \in \mathcal{A}) = 1-\gamma$ for any $0 < \gamma < 1$, $D_{KL}(D_\gamma \| D_{\mathbb{Z},c,s})$ is bounded by

$$D_{KL}(D_\gamma \| D_{\mathbb{Z},c,s}) = \gamma + O(\gamma^2). \tag{31}$$

and so is $D_{KL}(P_{X_{1:r}} \| D_{\mathbb{Z},c,s})$.

4.3 Approximation error from polar sampling

The target discrete Gaussian distribution is tailcut to be $P_{X_{1:r}}$ which is exactly the distribution of r bits of Bernoullis. Let $P_{X_{1:r}^{1:N}}$ denote the distribution of N i.i.d. $X_{1:r}$. As discussed in Sect. 3, for properly chosen $0 < \beta < 0.5$, $N = 2^n, n \geq 1$ and the corresponding high- and low-entropy sets defined in (26) and (27), one can approximate $P_{X_{1:r}^{1:N}}$ in a level-by-level manner using PolarSampler$(\cdot)$ for r times giving rise to the produced distribution $Q_{X_{1:r}^{1:N}}$.

Recall it in Theorem 3 that an intermediate distribution Q' is introduced to analyse the KL divergence between P and Q. Likewise, for every $1 \leq k \leq r$ we introduce an intermediate distribution $Q'_{X_k^{1:N}}$ such that $X_k^{1:N} = U_k^{1:N} G_N$ and

$$U_k^{(i)} = \begin{cases} 0 \ w.p. \ \frac{1}{2} \\ 1 \ w.p. \ \frac{1}{2} \end{cases} \quad \text{if } i \in \mathcal{H}_{X_k|X_{1:k-1}},$$

$$U_k^{(i)} = \begin{cases} 0 \ w.p. \ P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}(0|x_{1:k-1}^{1:N}, u_k^{1:i-1}) \\ 1 \ w.p. \ P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}(1|x_{1:k-1}^{1:N}, u_k^{1:i-1}) \end{cases} \quad \text{if } i \in \mathcal{H}^c{}_{X_k|X_{1:k-1}},$$

where only the deterministic decisions for $U_k^{(i)}$ in $\mathcal{L}_{X_k|X_{1:k-1}}$ are replaced by random decisions. We can bound the KL divergence between $P_{X_{1:r}^{1:N}}$ and $Q'_{X_{1:r}^{1:N}}$ as

$$\begin{aligned} & D_{KL}\left(P_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}}\right) \\ & \overset{(a)}{=} D_{KL}\left(P_{U_{1:r}^{1:N}} \| Q'_{U_{1:r}^{1:N}}\right) \\ & \overset{(b)}{=} D_{KL}\left(P_{U_1^{1:N}} P_{U_2^{1:N}|U_1^{1:N}} \cdots P_{U_r^{1:N}|U_{1:r-1}^{1:N}} \| Q'_{U_1^{1:N}} Q'_{U_2^{1:N}|U_1^{1:N}} \cdots Q'_{U_r^{1:N}|U_{1:r-1}^{1:N}}\right) \\ & \overset{(c)}{=} \sum_{k=1}^{r} \sum_{i=1}^{N} D_{KL}\left(P_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}} \| Q'_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}}\right) \overset{(d)}{\leq} 2 \ln 2 \cdot r N 2^{-N^\beta}, \end{aligned} \tag{32}$$

where the equalities and inequalities are derived by (a) one-to-one mapping from $X_{1:r}^{1:N}$ to $U_{1:r}^{1:N}$; (b) the chain rule of joint distribution; (c) the chain rule of KL divergence; (d) Theorem 3. Likewise, we bound the KL divergence between $Q_{X_{1:r}^{1:N}}$ and $Q'_{X_{1:r}^{1:N}}$ as

$$\begin{aligned} D_{KL}(Q_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}}) &= D_{KL}(Q_{U_{1:r}^{1:N}} \| Q'_{U_{1:r}^{1:N}}) \\ &= \sum_{k=1}^{r} \sum_{i=1}^{N} D_{KL}(Q_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}} \| Q'_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}}) \overset{(e)}{\leq} \ln 2 \cdot r N 2^{-N^\beta}, \end{aligned} \tag{33}$$

where the inequality (e) is derived by Theorem 3. An explicit security analysis based on the KL divergence $D_{KL}(P_{X_{1:r}} \| D_{\mathbb{Z},c,s})$, $D_{KL}(P_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}})$ and $D_{KL}(Q_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}})$ will be given in the sequel.

Remark 1 According to the KL-based closeness, GaussianSampling$(\cdot)$ can arbitrarily approximate $D_{\mathbb{Z}^{1:N},c,s}$ for sufficiently large N and properly chosen β and r. We highlight that the multilevel polar sampler would be attractive in applications consuming many more than one discrete Gaussian samples. There are plenty of applications of this kind in lattice-based cryptography and the prominent one is FHE. In FHE, it is quite common that dimension N can be tens of thousands. Even for lattice signature schemes (e.g. BLISS, Falcon), $N = 512,\ 1024$ is quite common, plus one may generate a batch of samples (except for embedded devices).

5 Security analysis and parameter selection

5.1 Security analysis with KL divergence

Lemma 3 (Bounding success probability variations, [30]) *Let $\mathcal{E}^{\mathcal{P}}$ be an algorithm making at most q queries to an oracle sampling from a distribution $\mathcal{P}$ and returning a bit. Let $\epsilon \geq 0$, and $\mathcal{Q}$ be a distribution such that $D_{KL}(\mathcal{P}\|\mathcal{Q}) < \epsilon$. Let x (resp. y) denote the probability that $\mathcal{E}^{\mathcal{P}}$ (resp. $\mathcal{E}^{\mathcal{Q}}$) outputs 1. Then, $|x - y| \leq \sqrt{q\epsilon/2}$.*

Security argument [31] It can be concluded from Lemma 3 that if a scheme is λ-bit secure with oracle access to a perfect distribution $\mathcal{P}$ and the KL divergence between $\mathcal{P}$ and another distribution $\mathcal{Q}$ satisfies $D_{KL}(\mathcal{P}\|\mathcal{Q}) \leq 2^{-\lambda}$, then this scheme is also about λ-bit secure with oracle access to $\mathcal{Q}$. Note that this security argument holds only if $\mathcal{E}$ is a search problem but not a decisional one. The security argument based on KL divergence satisfies symmetry and triangle inequality though KL divergence itself does not (see [31, Sect. 3.2] for detail).

Consider that a scheme with access to a perfect distribution $D_{\mathbb{Z}^N,c,s}$ is λ-bit secure. Assume an adversary obtains N samples at each query. By the additivity of KL divergence and Eq. (31), we have $D_{KL}(D_{\mathbb{Z}^{1:N},c,s} \| P_{X_{1:r}^{1:N}}) \leq N(\gamma + \gamma^2)$. In order to achieve λ-bit security after the tailcut, we need to set $N(\gamma + \gamma^2) \approx 2^{-\lambda}$ by selecting $t \approx \sqrt{(\lambda + \log N)\ln 2/\pi}$. The number of levels needed is therefore $r = \lceil \log(2t \cdot s) \rceil$. As given in Sect. 4.3, the approximation error introduced by polar sampling is determined by both $D_{KL}(P_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}})$ and $D_{KL}(Q'_{X_{1:r}^{1:N}} \| P_{X_{1:r}^{1:N}})$ which are upper bounded as

$$D_{KL}(P_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}}) \leq 2\ln 2 \cdot rN2^{-N^\beta}, \quad D_{KL}(Q_{X_{1:r}^{1:N}} \| Q'_{X_{1:r}^{1:N}}) \leq \ln 2 \cdot rN2^{-N^\beta}.$$

In order to preserve λ-bit security after $P_{X_{1:r}^{1:N}}$ is replaced by $Q_{X_{1:r}^{1:N}}$, we need to select $n = \log N$ and β properly such that $2^{-2^{n\beta}+n+\log(r)+1} \approx 2^{-\lambda}$.

Figure 6 shows how the security level of interest is related to β in terms of different s and n. We can observe that the curves with the same n but different in s are quite close. It is understandable because the security is dependent on the approximation error as is almost independent of what the target distribution is if the proposed GaussianSampling$(\cdot)$ is used. We also find that to preserve $\lambda = 128$ bits of security, a larger n implies a smaller β. This is because larger n means deeper polarization and therefore smaller unpolarized set (i.e. smaller β). This observation is instructional in selecting parameters for GaussianSampling$(\cdot)$. At the implementation stage, the entropy consumption to produce $U_k^{(i)}$ are totally different for

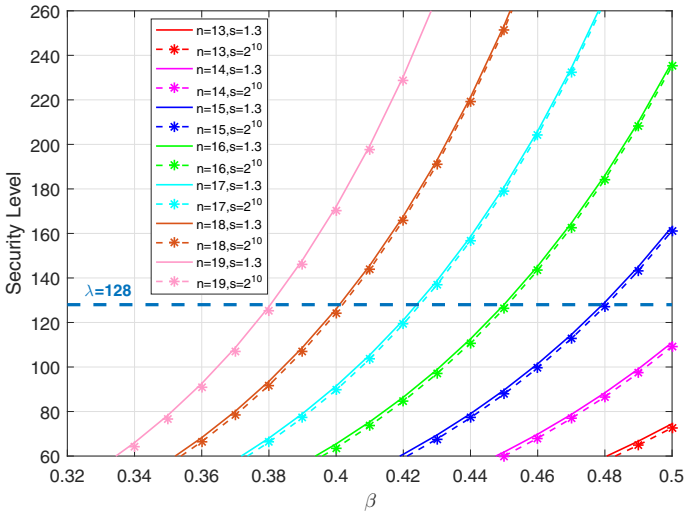

Fig. 6 Security level λ vs. β: $s = 1.3$ and $s = 2^{10}$

polarized and unpolarized set. Given optional choices of β and n to preserve λ-bit security, we suggest smaller β for less entropy consumption per sample.

5.2 Security analysis of tailcut and precision with Rényi divergence

The KL-based security analysis is a reminder about Rényi divergence (RD). However, the approximation error of the proposed sampler measured by Rényi divergence is not given in this work because how polarization phenomenon converges in the metric of RD is an open problem in the area of coding theory. Nonetheless, RD can still be used to analyse the tailcut and precision.

Definition 5 *(Rényi divergence)* Let P, Q be two distributions with supports $\mathcal{S}_P$ and $\mathcal{S}_Q$, respectively. Let $\mathcal{S}_P \subseteq \mathcal{S}_Q$. For $a \in (1, +\infty)$, we define the Rényi divergence of order a by

$$R_a(P\|Q) = \left(\sum_{x \in \mathcal{S}_P} \frac{P(x)^a}{Q(x)^{a-1}} \right)^{\frac{1}{a-1}}.$$

In [32], a sharper bound on security based on Rényi divergence is given under the assumption that the number of adversarial queries q to a λ-bit secure scheme is far less than 2^λ. Consider a cryptographic scheme with λ bits of security and the number of queries to $\mathcal{Q}$ satisfies $q \leq 2^{64}$. By the security argument in [32], this scheme is proved to lose at most one bit of security when $\mathcal{Q}$ is replaced by $\mathcal{Q}_\gamma$ provided that one of following conditions is satisfied.

(a) If $\mathcal{Q}_\gamma$ is the distribution under tailcut, then

$$\frac{\mathcal{Q}_\gamma}{\mathcal{Q}} \leq 1 + \gamma \text{ for } \gamma = \frac{1}{4q}.$$

(b) If $\mathcal{Q}_\gamma$ denote a distribution having the same support with $\mathcal{Q}$ subject to some relative error, it should be satisfied that

$$1-\gamma \leq \frac{\mathcal{Q}_\gamma}{Q} \leq 1+\gamma \text{ for } \frac{\gamma^2}{(1-\gamma)^{a+1}} \leq \frac{1}{4\lambda q}.$$

In the context of multilevel polar sampling, if $t \approx \sqrt{\frac{\ln 2(66+\log N)}{\pi}}$ and $r = \left\lceil \log \left(2s\sqrt{\frac{(66+\log N)\ln 2}{\pi}}\right)\right\rceil$, a $(\lambda+1)$-bit secure scheme will be at least λ-bit secure when the ideal distribution $D_{\mathbb{Z},c,s}$ is replaced by its tailcut $P_{X_{1:r}}$ for $\lambda = 128$, $q = 2^{64}$ and $a = 2\lambda$. Compared with the KL-based analysis of tailcut, Rényi divergence contributes to a smaller r by reducing the partitions by at most 1 level. Moreover, the security argument for relative error can be translated to $\gamma \leq 2^{-36.5}$ for $\lambda = 128$, $q = 2^{64}$ and $a = 2\lambda$ [32]. The precision requirement of polar sampler to achieve the target security is determined by the Bernoulli sampling step in formula (28) and (29). A practical approach to draw Bernoulli samples is to calculate the bias Q_γ subject to the relative precision provided. Then sample $q \in (0, 1)$ uniformly at random and yield 1 if $q < Q_\gamma$. As long as the relative precision provided for the LR recursions in Algorithm 2 and the biases computed in formula (28) and (29) is more than 36 bits, polar sampler can achieve $\lambda = 128$. In our application, it suffices to use double precision in the LR recursions which provides 52 bits of relative precision. It can also be simulated using fixed-point numbers of 64 bits of precision particularly in 64-bit architectures.

6 Complexity and comparison

6.1 A constant-time algorithm

Given a probabilistic or deterministic algorithm, we consider it to be constant-time if its execution time is independent of the sensitive part of its input and output [17]. Instead of making every operation finish exactly in a constant interval, we protect sensitive information from being recovered by timing-based side channel attacks. The proposed GaussianSampling(·) algorithm composes of multiple serially connected PolarSampler(·). We now study which part of PolarSampler(·) is constant-time and which part is not.

The input of PolarSampler(·) includes the block length N, the side information vector $y^{1:N}$, a precomputed bias table $\mathbf{c}$ and the corresponding high- and low-entropy sets $\mathcal{H}_{X|Y}$, $\mathcal{L}_{X|Y}$, meanwhile it yields a Bernoulli sample vector $x^{1:N}$. Normally, we do not expect to disclose either the Bernoulli biases or the output samples. Therefore, it makes sense to consider N and $y^{1:N}$ to be non-sensitive as they are irrelevant to what the target biases and output samples are. Table $\mathbf{c}$ and the two sets $\mathcal{H}_{X|Y}$, $\mathcal{L}_{X|Y}$ are sensitive as the Bernoulli biases $P(X=1|Y)$ for every $Y = y$ are stored in $\mathbf{c}$ and $\mathcal{H}_{X|Y}$, $\mathcal{L}_{X|Y}$ are also computed according to $P(X=1|Y)$.

PolarSampler(·) composes of four types of operations: (a) table lookup for $\mathbf{c}[y^{1:N}]$ given $y^{1:N}$; (b) recursive calculating the LRs by SC decoding; (c) probabilistic/deterministic sampling of 1 bit; (d) calculating $x^{1:N} = u^{1:N}G_N$. Whether these operations are relevant to the aforementioned sensitive information is listed in Table 1.

Firstly, the table lookup for $\mathbf{c}[y^{1:N}]$ may remind us of the cache-based attack breaking BLISS which exploits the weakness of CDT table search and the Bernoulli sampling with a precomputed table of exponential values [8]. Fortunately, this is not the case for PolarSampler(·). On one hand, the bias vector $\mathbf{c}[y^{1:N}]$ is indexed by the side information

Table 1 PolarSampler($\cdot$): Whether the execution time is relevant to sensitive information; relevant: ✓; irrelevant: ×; alternative for recursively calculating LRs: $*$

Operations	Positions in Algorithm 1	Bernoulli biases $\mathcal{H}_{X\|Y}$, $\mathcal{L}_{X\|Y}$	Precomputed table $\mathbf{c}$	Output samples
Table lookup	Line 2	×	×	×
Recursively calculating LRs	Line 4	×	✓	×
Division-free alternative of LR* calculations	Line 4	×	×	×
Probabilistic/deterministic sampling	Line 6,9,12	✓	×	×
$x^{1:N} = u^{1:N} G_N$	Line 14	×	×	×

$y^{1:N}$ and no binary search or search-with-guide-table method as in CDT sampling is needed. On the other hand, the Bernoulli sampler in BLISS leaks information about the yielded Bernoulli samples because of the conditional branching in the table lookup and bitwise sampling process. As a result, Bernoulli samples yielded faster (resp. slower) are more likely to be 0 s (resp. 1 s). But such conditional branches are not needed when searching for $\mathbf{c}[y^{1:N}]$ in $\mathbf{c}$ as long as table $\mathbf{c}$ is allocated with continuous memory. In this case one can easily find $\mathbf{c}[y^{1:N}]$ by moving the pointer to $\mathbf{c}$ by offset $y^{1:N}$. This operation is irrelevant to what the biases and the output are.

Secondly, if the block length is N, the SC decoding carries out exactly $\frac{N \log N}{2}$ LR calculations as in formula (13) along with exactly $\frac{N \log N}{2}$ LR calculations as in formula (14) regardless of what the input and output are. Concerns arise from the floating-point instructions and a comprehensive analysis is given as follows.

- If we assume floating-point instructions to be constant-time as in [17][6] or we make a constant-time transformation to those floating-point calculations (e.g., CTFP transformation [2] or replacement by fixed-point instructions), those LR calculations will be safe.
- If we only consider the floating-point division hard to be implemented in constant time as in [33, 39], a division-free alternative of the LR calculations of the same computational complexity is given in Appendix B and the above concern will be eliminated.
- Otherwise, the running time of LR calculations might be somehow relevant to $\mathbf{c}$. Recall it in Fig. 3 that the intermediate LRs are deduced from the rightmost column of LRs which are derived by $\mathbf{c}[y^{1:N}]$. It might take longer/shorter to finish an LR calculation given the two LRs of some specific values from last round of recursion. However, instead of observing a single LR calculation, an adversary has to guess $\mathbf{c}$ given the total running time of $N \log N$ LR calculations which may be impractical. Furthermore, the shuffled circuit makes the timing-based cryptanalysis even harder. As for the sensitive output, they are irrelevant to the running time taken to finish all the LR calculations. Only the N LR results in register LRReg[:][n] (i.e. the leftmost column LRs in Fig. 3) derived by the end of SC decoding and the succeeding deterministic/probabilistic sampling in line 6, 9 and 12 of Algorithm 1 determine the output samples, while how long it takes to calculate the intermediate LRs does not.

Thirdly, line 6, 9 and 12 of Algorithm 1 are non constant-time with respect to the input if no further measurements are taken. Sampling for the high- and low-entropy sets would be easy as it consumes only 1 bit of randomness for $\mathcal{H}_{X|Y}$ and 0 for $\mathcal{L}_{X|Y}$, whereas sampling for $\mathcal{H}^c_{X|Y} \backslash \mathcal{L}_{X|Y}$ is complicated and would take longer time. Therefore, the running time of line 6, 9 and 12 may reveal the proportion of $\mathcal{H}_{X|Y}$, $\mathcal{L}_{X|Y}$ and $\mathcal{H}^c_{X|Y} \backslash \mathcal{L}_{X|Y}$ which in turn discloses some information about the biases. As for the output, we consider it to be safe regardless of the running time of line 6, 9 and 12. The weak points are those floating-point comparisons in line 9 and 12. Generally speaking, comparing two approximate floating-point values would take longer, but it is equally likely to return *True* and *False* nonetheless. Therefore, it makes sense to consider this type of operations to be unsafe with respect to input but safe with respect to output.

Lastly, calculating $x^{1:N} = u^{1:N} G_N$ takes exactly $N \log N$ binary additions which is obviously constant-time.

To conclude, we claim that PolarSampler($\cdot$) is constant-time in the sense that its running time is irrelevant to the output samples. The same statement holds for the proposed integer

[6] A weaker notion dubbed "isochronous" is used instead of constant-time.

Gaussian sampling for two reasons if PolarSampler(·) is adopted as in Algorithm 4. Firstly, the number of levels r is determined by the width of the integer Gaussian distribution. Secondly, if no further measurements are taken, the floating-point implementation of PolarSampler(·) is non constant-time with respect to input.

6.2 Time complexity

The latest trend of DGS solutions is to expand a base sampler into one for arbitrary parameters. For example, the Knuth–Yao and CDT sampler can work as a base sampler to produce samples which are then combined into new samples with a relatively large standard deviation in a convolutional manner [26, 30]. Our Gaussian sampler is also eligible for such extension for potential speedup. The focus of this subsection is to compare the GaussianSampling(·) with other base samplers. Karmakar et al. [19] compared the time complexity of Knuth–Yao and CDT showing that the former can be made more time-saving. Therefore, it is fair to compare our Gaussian sampler only with Knuth–Yao and we used a non constant-time Knuth–Yao implemented in C++[7] as well as its constant-time version.[8] We also give the benchmarks of a prototype GaussianSampling(·) for different choices of parameter s in C++. [9] Note that we substitute probability recursion for LR recursion (see Appendix B) to avoid floating-point divisions.

The experiment is conducted on a PC with Ubuntu 18.04 and an Intel i9-9900K processor running at 3.60 GHz using one core. We use g++ to compile both Knuth–Yao and our implementation with compilation flag -Ofast enabled. For the benchmarks, we select $s \in \sqrt{2\pi} \cdot \{3, 8, 32, 256\}$ and a target security level $\lambda = 64$.[10] According to the KL-based security analysis in Sect. 5, we specify β to achieve 64 bits of security with respect to $N \in \{2^{13}, 2^{14}, 2^{15}\}$, and we select $r = \lceil \log(2st) \rceil$ where $t \approx \sqrt{(\lambda + \log N) \ln 2/\pi}$. We assume that the adversary obtains N integer samples for each query to the sampling algorithm. The simulation results are shown in Table 2. Firstly, our Gaussian sampler always outperforms Knuth–Yao in speed with respect to the above setting. Secondly, Knuth–Yao slows down almost linearly as 2^r grows while GaussianSampling(·) still provides a competitive speed. This advantage stems from the binary partition of the integers. Thirdly, GaussianSampling(·) shows modest speed reduction as the block length increases from 2^{13} to 2^{15}. This doesn't contradict the asymptotic information optimality which implies less randomness consumption per sample for larger N. The polarization effect can reduce the consumption of randomness and contribute to the speed. However, the overall running time, in the current implementation, is dominated by the floating-point recursions in Fig. 3 with complexity $O(N \log N)$. In the literature, practical boosting approaches for the butterfly circuit include semi-parallel design [21] and pruned SC decoding [1, 38] giving computational complexity up to $O(\log \log N)$.

6.3 Memory cost

At the construction stage, we need to store a table $\mathbf{c}$ of biases, i.e., $P(x_k = 1 | x_{1:k-1})$ for $1 \le k \le r$. The table consists of $2^r - 1$ elements for the overall r levels. The biases are

[7] https://github.com/AaronHall4/BKW-Algorithm.

[8] https://github.com/jnortiz/HIBE-Gaussian-Sampling.

[9] https://github.com/jwangit/polarsampler.

[10] GaussianSampling(·) is extendable to other security levels (e.g. 128,256) in our double precision setting as discussed in Sect. 5.2.

Table 2 Comparison between the GaussianSampling(·) and Knuth–Yao

2^r	s	Knuth–Yao (samples/s)		GaussianSampling(·) (samples/s)		
		Constant-time	Non constant-time	$N = 2^{13}$	$N = 2^{14}$	$N = 2^{15}$
2^6	$3\sqrt{2\pi}$	2.809E5/s	3.876E5/s	$\beta = 0.487$, 1.333E6/s	$\beta = 0.4535$, 1.283E6/s	$\beta = 0.4244$, 1.168E6/s
2^7	$8\sqrt{2\pi}$	1.172E5/s	2.212E5/s	$\beta = 0.4876$, 1.194E6/s	$\beta = 0.454$, 1.097E6/s	$\beta = 0.425$, 1.010E6/s
2^9	$32\sqrt{2\pi}$	3.255E4/s	6.017E4/s	$\beta = 0.488$, 0.960E6/s	$\beta = 0.4544$, 0.861E6/s	$\beta = 0.4252$, 0.792E6/s
2^{12}	$256\sqrt{2\pi}$	4.464E3/s	6.760E3/s	$\beta = 0.4885$, 0.680E6/s	$\beta = 0.455$, 0.621E6/s	$\beta = 0.4257$, 0.572E6/s

stored in natural order of $X_{1:k-1}$ such that once the samples $x_{1:k-1}$ for the preceding $k-1$ levels are ready we can find the associated biases in $\mathbf{c}$ by moving the pointer by offset $x_{1:k-1}$.

6.4 Asymptotic comparison

We also give in Table 3 an asymptotic comparison of entropy consumption, computational and storage complexity between GaussianSampling$(\cdot)$ and other existing samplers, e.g. the binary sampling algorithm [11], constant-time CDT [16], constant-time Knuth–Yao sampler [19].

The overall running time of GaussianSampling$(\cdot)$ depends on the SC decoding (LR recursions) and Bernoulli sampling (entropy consumption). As indicated in Corollary 1, the fraction of unpolarized set scales as $N^{-\frac{1}{\mu}}$. When $N \to \infty$, the average entropy consumption approaches $H(X)$ which is the Shannon's entropy of the target distribution X.

For binary sampling in BLISS [11], one sample requires entropy consumption of approximately $6 + 3 \times \log(s)$. If we use a full-table access CDT for the base sampler and use a full-table access Bernoulli sampler, the computational complexity would be $O(\log(t \times s_0))$ (s_0: the parameter of a base sampler) for table lookup plus some constant number of integer arithmetic and the entropy cost will be much greater than $6+3 \times \log(s)$. Falcon uses bimodal Gaussian and rejection sampling which should have similar complexity to binary sampling.

The full-table access CDT [16] has a computational complexity of $O(\log(t \times s))$ and requires a storage of $O(\lambda \times t \times s)$. The constant-time Knuth–Yao in [19], which is a variant of standard Knuth–Yao in [12] and performs totally different types of operations, has a computational complexity of $O(\log(t \times s))$ for Boolean function evaluations with λ random bits of input and its entropy consumption is $O(\lambda)$. It requires large programmable memory to store $O(\log(t * s))$ Boolean functions.

In conclusion, we claim our multilevel polar sampler to achieve the information-theoretic optimality (i.e., asymptotically optimal entropy consumption) which compares favorably with other samplers. For most sampling methods, the overall speed depends on computational complexity and randomness generation (e.g. producing Bernoulli samples in the binary sampling method). The computational complexity of our sampler is less attractive due to the $\log(N)$ factor but (a) multilevel polar sampler saves the time of producing randomness which is not considered as computational complexity but entropy consumption (b) our experiments in Table 2 show that multilevel polar sampler is much faster than a standard Knuth–Yao [12]. In addition, there is room to improve the computational efficiency seeing that the state-of-the-art SC decoding achieves a per-bit complexity of $O(\log \log N)$ [38].

7 Conclusions and future work

The polar sampler and its multilevel application for DGS is efficient, application-independent and constant-time. Our algorithm is effective in the case that a large number of samples from a certain distribution, e.g., integer Gaussian, are required. The optimization of entropy consumption stems from the polarization process in which the randomness moves to the high-entropy set. For fixed parameters, the construction stage is prepared offline and the implementation stage is carried out online. The floating-point implementation given in this work is constant-time in the sense that its running time is independent of output samples. KL and Rényi divergence are used for security analysis, precision analysis and parameter selection. Since the Rényi divergence-based analysis of polar coding is still an open problem

Table 3 Comparison of entropy, computational and storage complexity between multilevel polar sampler and existing samplers (λ: precision; $t * s$: tailcut; H/N: the fraction of high entropy set; $H(X)$: the entropy of X; μ is a constant bounded as $3.579 \leq \mu \leq 4.714$ according to Lemma 1)

Schemes	Computational complexity/sample	Entropy consumption/sample	storage
Multilevel polar sampler	$O(t \times s \times \log N)$ floating point[a]	$O(\lambda \times N^{-\frac{1}{\mu}} + H/N) \rightarrow H(X)$	$O(\lambda \times t \times s)$
Binary sampling [11]	$O(\log(s))$ table lookup	$\approx 6 + 3 \times \log(s)$	$O(\lambda \times \log(2.4 \times t \times s^2))$
Constant-time CDT [16]	$O(\log(t \times s))$ table lookup	λ	$O(\lambda \times t \times s)$
variant of Knuth–Yao [19]	$O(\log(t \times s))$ Boolean functions	λ	$O(\log(t \times s))$ Boolean functions

[a]Or $O(t \times s \times \log\log N)$ floating point operations if the state-of-the-art SC decoding in [38] is used

by now, it deserves more efforts to give a complete Rényi divergence-based analysis for our sampler and to carry out potential efficiency improvement.

In this paper, we only use the basic 2×2 kernel, whose finite-length performance is not the best. Optimizing finite-length performance using other kernels of polar codes as well as adapting the pruned/semi-parallel SC decoding are left to future work.

Funding This work was funded by the UK Engineering and Physical Sciences Research Council (Grant EP/S021043/1).

Data availability All data generated or analysed during this study are included in this published article.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

Appendix

A KL divergence for the low-entropy set

For $i \in \mathcal{L}_{X_k|X_{1:k-1}}$, Q' and Q follow the distribution respectively as

$$Q'\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right) = P\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right)$$
$$Q\left(\bar{u}_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right) = 1 \text{ for } \bar{u}_k^{(i)} = \arg\max_{u\in\{0,1\}} P_{U_k^{(i)}|X_{1:k-1}^{1:N}, U_k^{1:i-1}}\left(u|x_{1:k-1}^{1:N}, u_k^{1:i-1}\right).$$

By definition of KL divergence, we have

$$\begin{aligned} &D_{KL}\left(Q_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}} \| Q'_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}}\right) \\ &= \sum_{u_k^{1:i-1}, u_{1:k-1}^{1:N}} Q'\left(u_k^{1:i-1}, u_{1:k-1}^{1:N}\right)[-1 \cdot \log Q'\left(\bar{u}_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right) \\ &\quad - 0 \cdot \log\left(1 - Q'\left(\bar{u}_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right)\right) + (0 \log 0 + 1 \log 1)]. \end{aligned}$$

By definition Shannon entropy,

$$
\begin{aligned}
&H_P\left(U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}\right)\\
&= -\sum_{u_k^{1:i-1}, u_{1:k-1}^{1:N}} Q'\left(u_k^{1:i-1}, u_{1:k-1}^{1:N}\right) \sum_{u_k^{(i)}} Q'\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right)\\
&\quad \times \log Q'\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right),
\end{aligned}
$$

for $i \in \mathcal{L}_{X_k|X_{1:k-1}}$. For $0.5 \le Q'(\bar{u}_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}) < 1$ which is easily satisfied by choosing the low-entropy set $\mathcal{L}_{X_k|X_{1:k-1}}$ properly, we can prove that

$$
\begin{aligned}
&\sum_{u_k^{(i)} \in \{\bar{u}_k^{(i)}, 1-\bar{u}_k^{(i)}\}} -Q'\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right) \log Q'\left(u_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right)\\
&\ge -\log Q'\left(\bar{u}_k^{(i)}|u_k^{1:i-1}, u_{1:k-1}^{1:N}\right),
\end{aligned}
$$

and $D_{KL}\left(Q_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}} \| Q'_{U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{N}}\right) \le H_P\left(U_k^{(i)}|U_k^{1:i-1}, U_{1:k-1}^{1:N}\right)$.

B A division-free alternative for LR calculations

As mentioned in Footnote 5, the LR recursion can be replaced by transition probability recursion which has the same computational complexity $O(N \log N)$ as well ([3, Proposition 3]). In the prototype implementation referred to Footnote 9, we give both LR recursion (i.e. polar::recursivelyCalcLR) and probability recursion (i.e. polar::recursivelyCalcP) and the benchmarks are derived by the latter. Since LR is defined as the ratio of transition probabilities, recursively calculating LRs and transition probabilities are equivalent and interchangeable. Specifically, instead of defining an LR array of dimension $N \times (n+1)$ as in Fig. 3, we define two probability arrays PReg_0 and PReg_1 of the same size $N \times (n+1)$. The notion of branch and phase will be the same as used in an LR array. Then the LR recursions can be replaced by the probability recursions as follows. To initiate, the rightmost column of the two probability arrays are $\mathrm{PReg}_0[\langle 1, \psi\rangle][0] = 1 - \mathbf{c}[y^{(\psi+1)}]$ and $\mathrm{PReg}_1[\langle 1, \psi\rangle][0] = \mathbf{c}[y^{(\psi+1)}]$, respectively where $0 \le \psi \le 2^n - 1$, $\mathbf{c}$ is the bias table and $y^{(\psi+1)}$ is the $(\psi+1)$-th coordinate of side information vector $y^{1:N}$.

Then, for $0 < m \le n$, we update PReg_0 and PReg_1 by

$$
\begin{aligned}
\mathrm{PReg}_0[\langle 2\kappa - 1, \psi\rangle][m] &= \frac{1}{2}\mathrm{PReg}_0[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_0[\langle \kappa, 2\psi+1\rangle][m-1]\\
&\quad + \frac{1}{2}\mathrm{PReg}_1[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_1[\langle \kappa, 2\psi+1\rangle][m-1],
\end{aligned} \tag{34}
$$

$$
\begin{aligned}
\mathrm{PReg}_1[\langle 2\kappa - 1, \psi\rangle][m] &= \frac{1}{2}\mathrm{PReg}_1[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_0[\langle \kappa, 2\psi+1\rangle][m-1]\\
&\quad + \frac{1}{2}\mathrm{PReg}_0[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_1[\langle \kappa, 2\psi+1\rangle][m-1],
\end{aligned} \tag{35}
$$

$$\mathrm{PRReg}_0[\langle 2\kappa, \psi\rangle][m] = \frac{1}{2}\mathrm{PReg}_{\mathrm{temp}}[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_0[\langle \kappa, 2\psi+1\rangle][m-1], \tag{36}$$

$$\mathrm{PRReg}_1[\langle 2\kappa, \psi\rangle][m] = \frac{1}{2}\mathrm{PReg}_{\mathrm{temp}\oplus 1}[\langle \kappa, 2\psi\rangle][m-1] \cdot \mathrm{PReg}_1[\langle \kappa, 2\psi+1\rangle][m-1], \tag{37}$$

where $\mathrm{temp} = \mathrm{UReg}[\langle 2\kappa - 1, \psi\rangle][m]$ and $\oplus$ is XOR operation. Formula (34), (35), (36) and (37) will substitute for the LR calculations in formula (13) and (14) respectively and line 5 and 6 in Algorithm 2 will be adapted accordingly. After the N probabilities in the leftmost column of PReg_0 and PReg_1 are derived, the probabilistic/deterministic sampling in line 9 and 12 of Algorithm 1 will be replaced by $\mathsf{UReg}[i][n] = \mathsf{PReg}_0[i][n] < \mathsf{PReg}_1[i][n]$ and $\mathsf{UReg}[i][n] = \mathsf{Uniform}() < \mathsf{PReg}_1[i][n]$, respectively.

References

1. Alamdar-Yazdi A., Kschischang F.R.: A simplified successive-cancellation decoder for polar codes. IEEE Commun. Lett. **15**(12), 1378–1380 (2011).
2. Andrysco M., Nötzli A., Brown F., Jhala R., Stefan D.: Towards verified, constant-time floating point operations. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1369–1382. CCS '18. Association for Computing Machinery, New York (2018).
3. Arıkan E.: Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. IEEE Trans. Inf. Theory **55**(7), 3051–3073 (2009).
4. Arıkan E.: Source polarization. In: 2010 IEEE International Symposium on Information Theory, pp. 899–903 (2010).
5. Bai S., Lepoint T., Roux-Langlois A., Sakzad A., Stehlé D., Steinfeld R.: Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. J. Cryptol. **31**(2), 610–640 (2018).
6. Barthe G., Belaïd S., Espitau T., Fouque P.A., Rossi M., Tibouchi M.: Galactics: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2147–2164. CCS '19. Association for Computing Machinery (2019).
7. Box G.E.P., Muller M.E.: A note on the generation of random normal deviates. Ann. Math. Stat. **29**(2), 610–611 (1958).
8. Bruinderink L.G., Hülsing A., Lange T., Yarom Y.: Flush, Gauss, and reload—a cache attack on the BLISS lattice-based signature scheme. In: International Conference on Cryptographic Hardware and Embedded Systems, pp. 323–345. Springer (2016)
9. Cover T.M., Thomas J.A.: Elements of Information Theory. Wiley, Hoboken (2012).
10. Devroye L.: Sample-based non-uniform random variate generation. In: Proceedings of the 18th Conference on Winter Simulation, pp. 260–265. ACM (1986)
11. Ducas L., Durmus A., Lepoint T., Lyubashevsky V.: Lattice signatures and bimodal Gaussians. In: Annual Cryptology Conference, pp. 40–56. Springer (2013)
12. Dwarakanath N.C., Galbraith S.D.: Sampling from discret Gaussians for lattice-based cryptography on a constrained device. Appl. Algebra Eng. Commun. Comput. **25**(3), 159–180 (2014).
13. Espitau T., Fouque P.A., Gérard B., Tibouchi M.: Side-channel attacks on BLISS lattice-based signatures: exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1857–1874. CCS '17, Association for Computing Machinery (2017).
14. Gentry C., Peikert C., Vaikuntanathan V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 197–206. ACM (2008).
15. Honda J., Yamamoto H.: Polar coding without alphabet extension for asymmetric models. IEEE Trans. Inf. Theory **59**(12), 7829–7838 (2013).
16. Howe J., Khalid A., Rafferty C., Regazzoni F., O'Neill M.: On practical discret Gaussian samplers for lattice-based cryptography. IEEE Trans. Comput. **67**(3), 322–334 (2016).
17. Howe J., Prest T., Ricosset T., Rossi M.: Isochronous Gaussian sampling: from inception to implementation. In: Ding J., Tillich J.P. (eds.) Post-Quantum Cryptography, pp. 53–71. Springer, Cham (2020).

18. Hülsing A., Lange T., Smeets K.: Rounded Gaussians: fast and secure constant-time sampling for lattice-based crypto. In: Public-Key Cryptography—PKC 2018—21st IACR International Conference on Practice and Theory of Public-Key Cryptography. Proceedings, pp. 728–757. Springer (2018).
19. Karmakar A., Roy S.S., Reparaz O., Vercauteren F., Verbauwhede I.: Constant-time discret Gaussian sampling. IEEE Trans. Comput. **67**(11), 1561–1571 (2018).
20. Knuth D.: The complexity of nonuniform random number generation. In: Traub J.F. (ed.) Algorithm and Complexity, New Directions and Results, pp. 357–428. Academic Press, Cambridge (1976).
21. Leroux C., Raymond A.J., Sarkis G., Gross W.J.: A semi-parallel successive-cancellation decoder for polar codes. IEEE Trans. Signal Process. **61**(2), 289–299 (2013).
22. Lyubashevsky V., Peikert C., Regev O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 1–23. Springer (2010).
23. Marsaglia G., Tsang W.W., et al.: The Ziggurat method for generating random variables. J. Stat. Softw. **5**(8), 1–7 (2000).
24. Micciancio D., Peikert C.: Hardness of SIS and LWE with small parameters. In: Canetti R., Garay J.A. (eds.) Advances in Cryptology-CRYPTO 2013, pp. 21–39. Springer, Berlin (2013).
25. Micciancio D., Regev O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007).
26. Micciancio D., Walter M.: Gaussian sampling over the integers: efficient, generic, constant-time. In: Annual International Cryptology Conference, pp. 455–485. Springer (2017).
27. Mondelli M., Hashemi S.A., Cioffi J.M., Goldsmith A.: Sublinear latency for simplified successive cancellation decoding of polar codes. IEEE Trans. Wirel. Commun. **20**(1), 18–27 (2021).
28. Peikert C.: An efficient and parallel Gaussian sampler for lattices. In: Annual Cryptology Conference, pp. 80–97. Springer (2010).
29. Pessl P., Bruinderink L.G., Yarom Y.: To BLISS-B or not to be: attacking strong Swan's implementation of post-quantum signatures. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, p. 1843–1855. CCS '17, Association for Computing Machinery (2017).
30. Pöppelmann T., Ducas L., Güneysu T.: Enhanced lattice-based signatures on reconfigurable hardware. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 353–370. Springer (2014).
31. Prest T.: Gaussian sampling in lattice-based cryptography. Ph.D. thesis, École Normale Supérieure (2015).
32. Prest T.: Sharper bounds in lattice-based cryptography using the Rényi divergence. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 347–374. Springer (2017).
33. Prest T., Ricosset T., Rossi M.: Simple, fast and constant-time Gaussian sampling over the integers for falcon. In: Tech. rep., Second PQC Standardization Conference. https://csrc.nist.gov/Presentations/2019/simple-fast-and-constant-time-gaussian (2019).
34. Regev O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, pp. 84–93. STOC '05, ACM (2005).
35. Saarinen M.J.O.: Arithmetic coding and blinding countermeasures for lattice signatures. J. Cryptogr. Eng. **8**(1), 71–84 (2018).
36. Tal I., Vardy A.: How to construct polar codes. IEEE Trans. Inf. Theory **59**(10), 6562–6582 (2013).
37. Tal I., Vardy A.: List decoding of polar codes. IEEE Trans. Inf. Theory **61**(5), 2213–2226 (2015).
38. Wang H.P., Duursma I.M.: Log-logarithmic time pruned polar coding. IEEE Trans. Inf. Theory **67**(3), 1509–1521 (2021).
39. Zhao R.K., Steinfeld R., Sakzad A.: Facct: fast, compact, and constant-time discret Gaussian sampler over integers. IEEE Trans. Comput. **69**(1), 126–137 (2019).