

PolarLAC

Polar code-enhanced **LA**ttice-based **C**ryptosystem

Principal Submitter: Xianhui Lu^{1,2}, +86-15010131069, luxianhui@iie.ac.cn

Contributors:

- **Design:**

Jiabo Wang^{1,2}, wangjiabo@iie.ac.cn

Dongshu Cai^{1,2}, caidongshu@iie.ac.cn

Yijian Liu^{1,2}, liuyijian@iie.ac.cn

Yamin Liu³, liuyamin3@huawei.com

- **Analysis:**

Lei Bi^{1,2}, bilei@iie.ac.cn

Dingding Jia^{1,2}, jiadingding@iie.ac.cn

Jingnan He^{1,2}, hejingnan@iie.ac.cn

- **Principal Implementation:** Ying Liu^{1,2}, liuying@iie.ac.cn

Ziyao Liu^{1,2}, liuziyao@iie.ac.cn

Yu Zhang^{1,2}, zhangyu1999@iie.ac.cn

- **Additional Implementation:** Bo Pang⁴, pangb21@sdc.icbc.com.cn

Organizations:

1. State Key Laboratory of Cyberspace Security Defense,

Institute of Information Engineering, CAS

2. University of Chinese Academy of Sciences

3. Huawei Technologies

4. Industrial and Commercial Bank of China

Owner:

Xianhui Lu

Postal Address:

No. 19, Shu Cun Road, Beijing 100093, China

Contact:

wangjiabo@iie.ac.cn, +86-17788217158

Table of Contents

1	Introduction	1
1.1	Relation to / Improvement upon LAC	1
2	Preliminaries	1
2.1	Notations	2
2.2	Distributions	2
2.3	Module Learning with Errors	3
2.4	Polar Codes	4
3	Design Rationale	6
3.1	Algebraic Structures	7
3.2	Modulus	7
3.3	Hardness Assumption	7
3.4	Bit-Level Distributions	8
3.5	Correlation of Decryption Errors	8
4	Description of the Cryptosystems	8
4.1	Public Key Encryption	8
4.2	Key Encapsulation Mechanism	11
5	Parameter Choices	13
5.1	Modulus	13
5.2	Distributions	14
5.3	Data Compression	15
5.4	Spectral Rejection Sampling	17
5.5	Recommended Parameter Categories	19
6	Decryption Failure Rate Analysis	19
6.1	The Thorn Structures of Polynomial Convolution	20
6.2	A Degraded Channel Model	20
6.3	PolarLAC under Existing DFR Attacks	23
6.4	PolarLAC under a Novel DFR Attack	27
6.5	Alternative Rejection Sampling via Autocorrelation Polynomial	29
7	Security Analysis	30
7.1	Theoretical Security	30
7.2	Practical Security	31
8	Implementation and Performance	32
8.1	Polynomial Multiplication	32
8.2	Integer FFT for Spectral Rejection	34
8.3	Efficient Sampling	35
8.4	Computational Performance	35
9	Features	38

List of Figures

1	Channel Polarization of LAC-128 in NIST round 2.	5
2	SC decoding process	6
3	Frequency spectrum: average-case vs. under cca attack	18
4	The distribution of the magnitude of $\text{FFT}(e \cdot r)$	20
5	P.D.F. as a function of l_2 -norm: e_{real} v.s. e_{vir} v.s. independence assumption.	22
6	Power spectrum of Type-1 and Type-2 with $C_{\text{pre}} = 2^{64}$ for PolarLAC-128	24
7	C_{pre} v.s. (L_0, L_1) for PolarLAC-128	25
8	The magnitude of $\text{FFT}_1(e)$ v.s. L_1 for PolarLAC-128	25
9	Type-2 vs Type-S	26

List of Tables

1	Fixed zero-selector parameters for public-key compression	16
2	Threshold values of T_0, T and rejection rates.	18
3	Parameter Sets of PolarLAC.KEM.CCA	19
4	Capability of error correction of PolarLAC under a resource exhaustion attack of Type-S (10^5 iterations).	27
5	Capability of error correction of PolarLAC under a resource exhaustion attack from frequency domain (10^5 iterations)	29
6	Partially expanded NTT structure for the MLWE parameter sets	33
7	Computational performance of the official SM3-based C and AVX2 implementations of PolarLAC.PKE.CPA.	36
8	Computational performance of the official SM3-based C and AVX2 implementations of PolarLAC.KEM.CCA.	36
9	Computational Performance of the SHAKE-based C and AVX2 Implementations of PolarLAC.PKE.CPA	37
10	Computational Performance of the SHAKE-based C and AVX2 Implementations of PolarLAC.KEM.CCA	37
11	Computational performance of the default SM3-based C and ARM/SVE implementations of PolarLAC.KEM.CCA (median ns/op).	38
12	Computational performance of the SHAKE-based C and ARM/SVE implementations of PolarLAC.KEM.CCA (median ns/op).	38

1 Introduction

In this document we introduce **PolarLAC** (Polar Codes Enhanced Lattice Based Cryptosystem), which comprises two public key cryptographic primitives based on the module learning with errors (MLWE):

- PolarLAC.PKE.CPA: an IND-CPA secure public key encryption scheme;
- PolarLAC.KEM.CCA: an IND-CCA secure key encapsulation mechanism, which is obtained by applying a variant of the FO transformation [23] to PolarLAC.PKE.CPA.

1.1 Relation to / Improvement upon LAC

To reduce the key and ciphertext sizes of lattice-based encryption while keep reasonable security strength, the most intuitive approach is to decrease the modulus size. In 2017, Lu et al. presented a bold attempt to use byte-level modulus, namely, $q = 251$, in the design of LAC [28], which was the smallest one among lattice-based candidates. LAC exhibited competitive performance and entered the second round of NIST PQC project [29], and also won the first prize during the first national algorithm design competition of China.

A limitation of LAC lies in its inaccurate decryption failure rate estimation. This issue is further exacerbated under chosen-ciphertext attacks (CCA), where adversaries can amplify the impact of inherent noise correlation via deliberate ciphertext pattern selection [22,15]. Leveraging a set of newly proposed techniques, **PolarLAC** overcomes the aforementioned limitation of LAC while preserving its advantage of small moduli. Specifically:

- Modulus: In **PolarLAC**, we choose $q = 257$ as a better trade-off between security, bandwidth, and implementation efficiency. Compared with $q = 251$ in LAC, the modulus $q = 257$ is NTT-friendly. Although this does not lead to a complete NTT decomposition for high-dimensional rings, it naturally supports an incomplete NTT, whose remaining schoolbook multiplications can be efficiently handled by optimized Karatsuba multiplication.
- Module-LWE: We construct **PolarLAC** based on module-LWE, which is widely conjectured to provide stronger security guarantees than RLWE. Furthermore, we find that the correlation between noise terms (e.g., $e_1(x)s(x)$) is significantly suppressed under MLWE. This is evidenced by significantly much lower decryption failure rates against adversaries with identical computational budgets.
- Polar codes: Error correction is critical to ensure a competitive decryption failure rate when we select small moduli and a code rate of $1/2$. We select polar codes after a comprehensive comparison among error-correcting codes of all kinds. It provides remarkable decryption failure rates thanks to the successive cancellation decoding which makes use of the soft information of the noise distribution. The hard-decision decoding of BCH misses this information. Moreover, polar codes gives a theoretical upper bound on decoding error rate. Additionally, it intrinsically supports constant-time implementation.
- Error correlation: A key improvement over LAC lies in securing a sufficiently low decryption failure rate in the CCA setting, where adversaries can exploit cross-correlations of error coefficients to inflate decryption failure risks. We characterize the origin of these correlations in the frequency domain [11], and deploy a spectral rejection sampling method to constrain noise and secret polynomials via their spectral norms. This effectively suppresses samples that would otherwise lead to excessive decryption failures. The upper bound on DFR is derived by devising a virtual and i.i.d. Gaussian noise which is stronger than the trimmed noise. The effectiveness of spectral rejection sampling, error correction are validated using DFR attacks [22], and particularly a even more powerful attack we devised. Details of the analysis are given in this proposal.

2 Preliminaries

In this section we first define several mathematical notations, then introduce backgrounds on lattices and error correction codes.

2.1 Notations

Vectors and Matrices. Vectors are denoted by bold lower-case characters, such as $\mathbf{a}$, and $\mathbf{a}^t$ denotes the transpose of $\mathbf{a}$.

An m -dimensional vector $\mathbf{a} = (a_0, \dots, a_{m-1})^t$, where the a_i 's are the components of $\mathbf{a}$ for $0 \leq i < m$.

For a scalar s and an m -dimensional vector $\mathbf{a}$, $s \cdot \mathbf{a}$ denotes that each component of $\mathbf{a}$ is multiplied by s , i.e., $s \cdot \mathbf{a} = (s \cdot a_0, \dots, s \cdot a_{m-1})^t$.

For an m -dimensional vector $\mathbf{a} = (a_0, \dots, a_{m-1})^t$ and a non-negative integer $l \leq m$, define $(\mathbf{a})_l = (a_0, \dots, a_{l-1})$.

Vectors of the same dimension can add component-wise, e.g., for two m -dimensional vectors $\mathbf{a} = (a_0, \dots, a_{m-1})^t$ and $\mathbf{b} = (b_0, \dots, b_{m-1})^t$, $\mathbf{a} + \mathbf{b} = (a_0 + b_0, \dots, a_{m-1} + b_{m-1})^t$.

Given $\mathbf{a}$ and $\mathcal{I} \subseteq [m]$, $\mathbf{a}_{\mathcal{I}}$ is the subvector consisting of the entries of $\mathbf{a}$ indexed by $\mathcal{I}$, ordered identically to $\mathbf{a}$.

Matrices are denoted by upper-case characters, such as $\mathbf{A}$, and $\mathbf{A}^t$ denotes the transpose of $\mathbf{A}$.

Norms. The length of vectors is measured with norms. For an m -dimensional vector $\mathbf{x} = (x_0, x_1, \dots, x_{m-1})^t$, its l_1 -norm is defined as $\|\mathbf{x}\|_1 = \sum_{i=0}^{m-1} |x_i|$; the l_2 -norm, also known as the Euclidean norm, is defined as $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=0}^{m-1} x_i^2}$, or solely denoted as $\|\mathbf{x}\|$; the infinity norm is defined as $\|\mathbf{x}\|_{\infty} = \max |x_i|$.

Arrows. For a set S , $x \xleftarrow{\$} S$ denotes that an element x is chosen from S uniformly at random. For a distribution D , $x \leftarrow D$ denotes that a random variable x is sampled according to the distribution D . For an algorithm $\mathbf{A}$, $y \leftarrow \mathbf{A}(x)$ denotes that y is assigned the output of $\mathbf{A}$ on input x .

Algebraic structures. Let $\mathbb{R}$ be real numbers, $\mathbb{Q}$ be rational numbers, and $\mathbb{Z}$ be integers. For an integer $q \geq 1$, let $\mathbb{Z}_q$ be the residue class ring modulo q , $\mathbb{Z}_q = \{0, \dots, q-1\}$, and the half of q as $\bar{q} = \lfloor \frac{q}{2} \rfloor$. Define the ring of integer polynomials modulo $x^n + 1$ as $R = \mathbb{Z}[x]/(x^n + 1)$ for an integer $n \geq 1$, and the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ denotes the polynomial ring modulo $x^n + 1$ where the coefficients are from $\mathbb{Z}_q$. The addition and multiplication of the elements in R_q are operated according to those of polynomials.

String operations. For two bit-strings $s_1, s_2 \in \{0, 1\}^*$, denote their concatenation as $s_1 \| s_2$. The length of a bit-string s is denoted as $|s|$. Sometimes, we treat bit-strings of length m as m -dimensional vectors. We use ϵ to denote an empty string.

2.2 Distributions

The Uniform Distribution. The uniform distribution over a set X is defined as $U(X)$. For example, the uniform distribution over R_q is $U(R_q)$.

The Gaussian Distribution. The normal Gaussian distribution is a continuous distribution with probability density function:

$$\rho_{\sigma}(x) = (\sqrt{2\pi}\sigma)^{-1} \exp(-\pi x^2 / 2\pi\sigma^2),$$

where σ is the standard deviation.

The complex Gaussian distribution has a probability density function of

$$\rho_{\sigma}(z) = (\pi\sigma^2)^{-1} \exp(-|z|^2 / \sigma^2), \quad z \in \mathbb{C}.$$

Ternary Distributions. In PolarLAC, three ternary distributions are used to generate the noise and secret terms. We define the three distributions $\Psi_{1/4}$, $\Psi_{3/8}$ and $\Psi_{11/64}$ as follows.

Definition 1 ($\Psi_{1/4}$). *It picks 0 with probability $\frac{3}{4}$, and ± 1 with probability $\frac{1}{8}$ according to the distribution $\Psi_{1/4}$. It can be realized by sampling $(a, b, c) \xleftarrow{\$} \{0, 1\}^3$, and yielding $(a - b) * c$. The mean value of $\Psi_{1/4}$ is 0 and the variance is $\frac{1}{4}$.*

Definition 2 ($\Psi_{3/8}$). *It picks 0 with probability $\frac{5}{8}$, and ± 1 with probability $\frac{3}{16}$ according to the distribution $\Psi_{3/8}$. It can be realized by sampling $(a, b, c, d) \xleftarrow{\$} \{0, 1\}^4$, and yielding $(a \wedge b) - (c \wedge d)$, where $\wedge$ is the AND operation. The mean value is 0 and the variance is $3/8$.*

Definition 3 ($\Psi_{11/64}$). *It picks 0 with probability $\frac{53}{64}$, and ± 1 with probability $\frac{11}{128}$ according to the distribution $\Psi_{11/64}$. It can be efficiently realized by the cumulative distribution table sampling. The mean value is 0 and the variance is $11/64$.*

Random Sampling. We define an abstract algorithm **Samp** as the procedure of sampling a random variable according to a distribution with a given seed:

$$x \leftarrow \text{Samp}(D; \text{seed}),$$

where D is a distribution, and **seed** is the random seed used to sample x .

2.3 Module Learning with Errors

Currently, most lattice-based public key encryption schemes and key exchange protocols are based on the learning with errors (LWE) assumption [33] and its variants. The proposed cryptosystems in PolarLAC are based on the learning with errors assumption over modules (MLWE) [26].

In PolarLAC, we denote by R and R_q the $2n$ -th cyclotomic rings $\mathbb{Z}/(x^n + 1)$ and $\mathbb{Z}_q/(x^n + 1)$, respectively. We use regular font to denote the polynomials in R or R_q ; the bold font is to denote vectors or matrices of polynomials.

Definition 4 (MLWE). *Let k be a positive integer and χ be a distribution. For arbitrary $\mathbf{s} \in R_q^k$, a module-LWE distribution $A_{q, \mathbf{s}, \chi}^M$ over $R_q^k \times R_q$ is $(\mathbf{a}, b)$ for some $\mathbf{a} \xleftarrow{\$} R_q^k$ and $b = \mathbf{a}^t \mathbf{s} + e$ with $e \leftarrow \chi$. The search problem is to recover $\mathbf{s}$ with $\text{poly}(n)$ many samples from $A_{q, \mathbf{s}, \chi}^M$ for some arbitrary $\mathbf{s} \in R_q^k$. The decisional problem is to distinguish $\text{poly}(n)$ many samples of $U(R_q^k \times R_q)$ from $A_{q, \mathbf{s}, \chi}^M$.*

Using standard techniques analogous to those in [30], one can prove that module-LWE, where each entry of $\mathbf{s}$ is independently and identically distributed following the distribution of e , i.e., $\mathbf{s} \leftarrow \chi$, remains hard at the cost of losing a certain number of samples.

In our practical scheme, we only utilize a subset of samples from MLWE instances without leaking any secret information. This design reduces the overall size of the scheme while incurring nearly no loss of security. We formally define a new problem, denoted MLWE_{Rej} , which performs filtering solely on MLWE samples without any auxiliary hints about the secret. Its only distinction from the standard MLWE problem is an extra rejection sampling step applied to the problem's input samples.

Definition 5 (MLWE_{Rej}). *Let Rej denote a rejection sampling filter over $R_q^k \times R_q$ with rejection rate M_{Rej} for the uniform distribution $U(R_q^k \times R_q)$.*

The search variant of MLWE_{Rej} receives polynomially many samples that are drawn from $A_{q, \mathbf{s}, \chi}^M$ and successfully pass through Rej , and requires recovering the secret vector $\mathbf{s}$.

The decision variant of MLWE_{Rej} receives polynomially many filter-passed samples, and requires distinguishing whether these samples originate from $A_{q, \mathbf{s}, \chi}^M$ or the uniform distribution $U(R_q^k \times R_q)$.

Standard cryptographic arguments imply that whenever $M_{\text{Rej}} \leq 1 - 1/\text{poly}(n)$, polynomially many valid MLWE_{Rej} instances can be reliably obtained from polynomially many standard MLWE samples, establishing a polynomial-time reduction from standard MLWE to MLWE_{Rej} . For our concrete construction, the adopted rejection rate M_{Rej} is constant-order ($1/O(1)$), which is far smaller than the threshold $1 - 1/\text{poly}(n)$.

2.4 Polar Codes

Polar codes, introduced by Arikan in 2009 [6], are the first class of channel codes that are provably capacity-achieving for any binary-input discrete memoryless channel (B-DMC). The fundamental principle underlying polar codes is channel polarization. Specifically, for a block length N , polar codes take N independent copies of a given channel W and apply a recursive polarization transform to construct N polarized channels. As N increases, these polarized channels tend to polarize towards two extremes: a subset becomes nearly noiseless with capacity approaching 1, while the remaining channels become almost completely noisy with capacity approaching 0. This polarization phenomenon enables reliable communication by transmitting information bits through the reliable polarized channels, while placing pre-shared frozen bits—e.g. all 0s—on the unreliable ones.

A Channel Model in KEMs. For KEMs where we expect to deliver 1 bit of message per symbol (i.e. a symbol means $\pm\lfloor q/2 \rfloor$), we can derive a channel $W : \{0, 1\} \rightarrow \mathcal{Y}$ which output a noisy signal $y \in \mathcal{Y}$ on some input $X = 0/1$. If there exists a permutation π on $\mathcal{Y}$ such that $P(y|x = 0) = P(\pi(y)|x = 1)$, the channel W is said to be symmetric. Many noises of common interest (e.g. additive Gaussian noise) give rise to symmetric channels. The Shannon's capacity of W is defined as below.

Definition 6 (Channel Capacity of a Binary Input Discrete Memoryless Symmetric (BDMS) channel). *Without loss of generality, we define the channel output $\mathcal{Y}$ over a discrete support. The channel capacity is defined as*

$$I(W) = \sum_{y \in \mathcal{Y}} \sum_{x \in \{0,1\}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)},$$

where $W(y|x)$ is the probability of the channel of outputting y conditional on input x .

Channel Polarization. Assume we want to use the BDMS channel W for $n = 2^k$ times for some integer k . Instead of sending $\mathbf{u} \in \{0, 1\}^n$ independently, we could apply a transform $\mathbf{G}$ on $\mathbf{u}$ where $\mathbf{G} = \mathbf{F}^{\otimes k}$, where $\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and $\mathbf{F}^{\otimes k}$ denotes the n -th Kronecker power of $\mathbf{F}$. We derive a *combined channel* as

$$W_n(\mathbf{y}|\mathbf{u}) = \prod_{i \in \{1, \dots, n\}} W(y_i | \mathbf{x}_i = (\mathbf{u}^t \mathbf{G})_i),$$

given that there is an one-to-one mapping between $\mathbf{x} = \mathbf{u}^t \mathbf{G}$ and $\mathbf{u}$.

Afterwards, we further split W_n into n *synthesized channel* as $W_n^{(i)} : \{0, 1\} \rightarrow \mathcal{Y} \times \{0, 1\}^{i-1}$ whose transition probability is defined by

$$W_n^{(i)}(\mathbf{y}, \mathbf{u}_{1:i-1} | u_i) = \sum_{u_{i+1:n} \in \{0,1\}^{n-i}} \frac{W_n(\mathbf{y}|\mathbf{u}) \cdot P(\mathbf{u})}{P(\mathbf{u}_i)}$$

Theorem 1 (Channel Polarization). *For any BDMS channel W , the synthesized channels $W_n^{(i)}$ polarize in the sense that, for any fixed $\delta \in (0, 1)$, as n goes to infinity through powers of 2, the fraction of indices $i \in \{1, \dots, n\}$ for which the Shannon's capacity $I(W_n^{(i)}) \in [0, \delta)$ goes to $1 - I(w)$.*

The reliability of $W_n^{(i)}$ will polarize simultaneously with $I(W_n^{(i)})$. In KEMs, we concern more about reliability than channel capacity. We give an example by heuristically apply polar codes to LAC in NIST round 2.

Example 1. The LAC-128 is endowed with a parameter set of $n = 512$, a central binomial distribution ψ of parameter 1, and a modulus of $q = 251$. If we heuristically deem the overall error term $r \cdot e - e_1 \cdot s + e_2$ to be Gaussian where, the $r, e, e_1, s \leftarrow \psi$. The channel capacity and bit error rate of the synthesized channel $W_n^{(i)}$ for $i = 1, \dots, n$ are depicted in Fig. 1. It is observed that the the overall channel capacity of LAC-128 is close to 1 bit per symbol. However, if we desire a block error rate below 2^{-128} , we are suggested to transmit messages using the most reliable synthesized channels. The union bound on the bit error rate of the selected synthesized channels gives the DFR. It is also observed that the bit error rate exhibits a layered distribution after several iterations of polarizations.

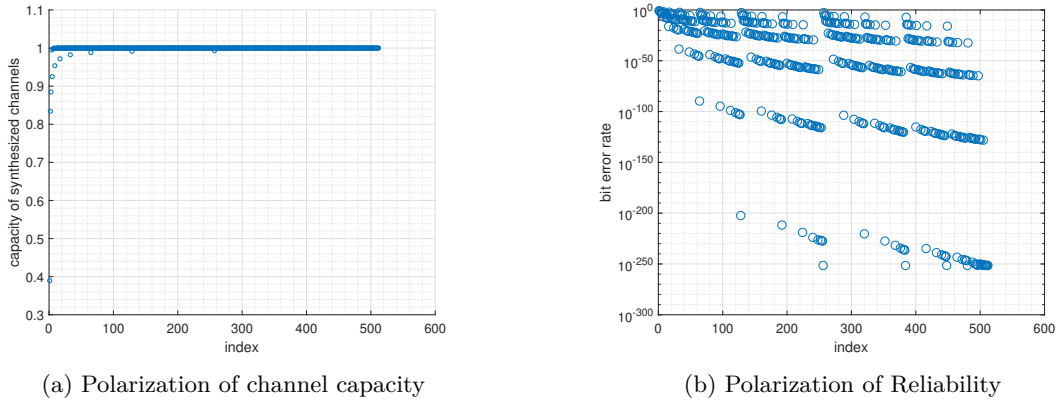


Fig. 1: Channel Polarization of LAC-128 in NIST round 2.

Polar Encoding. To encode a message from the message space $\{0, 1\}^l$ using polar codes, one needs to select the most reliable l synthesized channels whose indices forms a set $\mathcal{I} \subset \{1, \dots, n\}$, the so called *information set*. Its complementary set is called the *frozen set*, denoted by $\mathcal{F}$. The messages are assigned to $\mathbf{u}_{\mathcal{I}}$ while we make $\mathbf{u}_{\mathcal{F}} = \mathbf{0}$. Then polar encoding is done by $\mathbf{x} = \mathbf{u}^t G$. It takes polynomial time to calculate the reliability of all the synthesized channels [34] and it is done in a pre-computation phase.

Successive Cancellation Decoding and Block Error Rate. The most commonly used decoding algorithm for polar codes is Successive Cancellation (SC) decoding with a complexity of $O(n \log n)$ [6]. For any B-DMC W , the block error rate (BLER) under SC decoding is upper-bounded by:

$$\text{BLER} \leq \sum_{i \in \mathcal{I}} \text{BER}(W_N^{(i)}),$$

where $\mathcal{I}$ denotes the information set, and $\text{BER}(W_N^{(i)})$ is the bit error rate.

The SC decoding process is illustrated in Fig. 2. For a polar code of length n , the SC decoder receives the channel output $\mathbf{y}$. It first decodes u_1 based on $\mathbf{y}$, then decodes u_2 using $\mathbf{y}$ and the estimate of u_1 . More generally, the decoder estimates u_i based on $\mathbf{y}$ and the previous estimates $\hat{u}_1, \dots, \hat{u}_{i-1}$, until all bits $u_1, \dots, u_n$ are decoded. If the i -th bit is a frozen bit, its estimate is directly set as $\hat{u}_i = 0$; otherwise, if the i -th bit is an information bit, $\hat{u}_i$ is obtained according to a soft-decision decoding rule.

The detailed procedure is as follows.

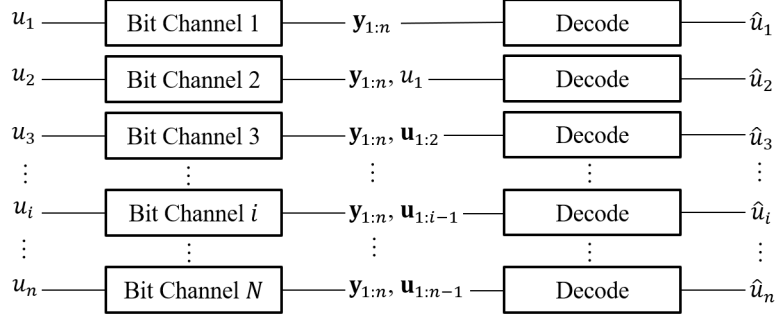


Fig. 2: SC decoding process

For $i \in \{1, 2, \dots, n\}$, after channel polarization, the transition probability of the i -th bit-channel is given by $W_n^{(i)}(\mathbf{y}, \mathbf{u}_{1:i-1} | u_i)$. The estimate $\hat{u}_i$ of bit u_i can be obtained sequentially by comparing the transition probabilities $W_n^{(i)}$ corresponding to $\hat{u}_i = 0$ and $\hat{u}_i = 1$.

For bit-channel index i ranging from 1 to n , the estimate of each bit is determined as

$$\hat{u}_i = \begin{cases} h_i(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1}), & i \in \mathcal{I}, \\ 0, & i \in \mathcal{F}, \end{cases}$$

where $i \in \mathcal{F}$ indicates that the i -th bit is a frozen bit, i.e., a bit predetermined by the legitimate parties, and hence it is directly set as $\hat{u}_i = 0$. In contrast, $i \in \mathcal{I}$ indicates that the i -th bit is an information bit, for which the decision function is given by

$$h_i(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1}) = \begin{cases} 0, & L_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1}) \geq 0, \\ 1, & L_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1}) < 0. \end{cases}$$

Here, the log-likelihood ratio (LLR) is defined as

$$L_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1}) \triangleq \ln \left(\frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1} | 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:i-1} | 1)} \right).$$

The LLRs can be recursively computed using the functions f and g as follows:

$$L_n^{(2i-1)}(\mathbf{y}, \hat{\mathbf{u}}_{1:2i-2}) = f \left(L_{n/2}^{(i)}(\mathbf{y}_{1:n/2}, \hat{\mathbf{u}}_{1:2i-2}^{\text{odd}} \oplus \hat{\mathbf{u}}_{1:2i-2}^{\text{even}}), L_{n/2}^{(i)}(\mathbf{y}_{n/2+1:n}, \hat{\mathbf{u}}_{1:2i-2}^{\text{even}}) \right),$$

$$L_n^{(2i)}(\mathbf{y}, \hat{\mathbf{u}}_{1:2i-1}) = g \left(L_{n/2}^{(i)}(\mathbf{y}_{1:n/2}, \hat{\mathbf{u}}_{1:2i-2}^{\text{odd}} \oplus \hat{\mathbf{u}}_{1:2i-2}^{\text{even}}), L_{n/2}^{(i)}(\mathbf{y}_{n/2+1:n}, \hat{\mathbf{u}}_{1:2i-2}^{\text{even}}), \hat{u}_{2i-1} \right)$$

where we denote by $\hat{\mathbf{u}}_{1:2i-2}^{\text{odd}}$ the entries associated with odd indices. The functions f and g are defined as

$$f(a, b) = \ln \left(\frac{1 + e^{a+b}}{e^a + e^b} \right) \approx \text{sign}(a)\text{sign}(b) \min\{|a|, |b|\},$$

$$g(a, b, u_s) = (-1)^{u_s} a + b.$$

Here, $\text{sign}(\cdot)$ denotes the sign function, $a, b \in \mathbb{R}$, and $u_s \in \{0, 1\}$.

3 Design Rationale

The main goal of the current design is to construct a light-weight lattice-based key encapsulation mechanism which balances reasonable security, compact bandwidth and implementation efficiency.

3.1 Algebraic Structures

While its precedent LAC is based on RLWE, we base PolarLAC on MLWE which is deemed to give a better balance between hardness assumption and efficiency. Moreover, dedicated attacks that exploiting the algebraic structure of ideal lattices are less likely to apply to MLWE than RLWE or NTRU. We define the module elements over the cyclotomic ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. To match the designated security strength of 128, 256, and 512 bits, we select the dimension of the module to be $k = 2$ and the dimension n of R_q to be 256, 512 and 1024. The selected n 's are powers of two, which is ideal for maximizing the efficiency of NTT.

3.2 Modulus

The choice of modulus is a pivotal factor in balancing security and performance, since the concrete security of (module-)LWE-based schemes is determined by the tuple (n, q, σ) , where n is the dimension, σ is the standard deviation, and q is the modulus. For fixed n and σ , reducing q generally increases the security margin against lattice reduction attacks and improves bandwidth efficiency. In this respect, choosing $q = 257$ provides a favorable trade-off: it keeps the modulus small while still supporting efficient NTT-based polynomial arithmetic. This advantage is not shared by nearby alternatives. The modulus $q = 251$ is NTT-unfriendly for power-of-two cyclotomic rings, because $q - 1 = 250$ does not contain a sufficiently large power-of-two factor to provide the required roots of unity for radix-2 NTT decomposition. The modulus $q = 256$ is also unsuitable for standard NTT arithmetic over a finite field, since it is not prime. By contrast, $q = 257$ is a Fermat prime with $q - 1 = 256$, and hence admits power-of-two roots of unity. Although this root structure may not allow a complete NTT decomposition for high-dimensional polynomial rings, it enables an efficient incomplete NTT. The use of an incomplete NTT is not merely a compromise. In highly optimized implementations, a fully decomposed NTT may leave very small terminal blocks, for which the overhead of additional decomposition, data movement, and function calls can outweigh the arithmetic savings. By stopping the transform earlier, the incomplete NTT leaves moderately sized base blocks that can be handled efficiently using optimized schoolbook, Karatsuba, or vectorized multiplication. Therefore, $q = 257$ simultaneously offers a small modulus for security and bandwidth, and a practical NTT-friendly structure that enables high-performance polynomial multiplication through incomplete decomposition. The only exception is for security category of 512, where we provide two parameter sets with $q = 257$ and $q = 769$, respectively. The set with $q = 257$ gives a decryption failure rate of 2^{-323} ; whereas the other set with $q = 769$ gives decryption failure rate of 2^{-544} and it supports partially expanded NTT. Both achieve the security strength of 512.

3.3 Hardness Assumption

Let $(\mathbf{a}, b) \in R_q^k \times R_q$ denote a standard MLWE sample instance. We formalize our modified variant below. Uniform sampling over R_{256} is significantly simpler than uniform sampling over R_{257} . To enable byte-sized representation of modulus $q = 257$, we therefore generate the public vector $\mathbf{a}$ over the module R_{256}^2 as a practical optimization. Additionally, to enable compact byte-wise storage of the term b , we deploy a second rejection sampling routine during the key generation phase of the PolarLAC scheme (with the PolarLAC-512* as the sole exception). This secondary filter enforces that the number of coefficients equal to 0 or 256 within b does not exceed a predefined threshold; if this bound is violated, the entire key generation process restarts. In this way, we are able to represent every coefficient of b with a single byte as a compression method, with a few bits appended to b for decompression.

These two rejection filters perturb the canonical module-LWE output distribution. To rigorously analyze the security of our modified sampling procedure, we introduce the hard problem module-LWE_{Rej}. We further prove a polynomial-time reduction from the standard module-LWE problem to module-LWE_{Rej}, which holds whenever the overall rejection rate is bounded above by $1 - 1/\text{poly}(n)$. The formal definition of module-LWE_{Rej} is provided in Definition 5.

3.4 Bit-Level Distributions

One caveat arises along with byte size modulus is that we need a precise control of the noises and secrets. The ratio of noise magnitude (measured with the standard deviation σ) over the modulus q is highly related to the security strength and the correctness of the proposed schemes. To achieve a robust security strength while keep a sufficiently low the decryption failure rate, we employ ternary distributions in PolarLAC.

3.5 Correlation of Decryption Errors

Correlations exist in the decryption errors of encryption schemes based on ring/module-LWE. It basically arises from the products of polynomials over rings. A CCA adversary could amplify the decryption failure rates at the cost of some computational effort. We found attacks of this kind could be identified in frequency domain. We eliminate those attacks in three steps. Firstly, we apply a technique, namely the spectral rejectoin sampling, which identified and filtered out the extremely bad secrets and noise terms such that they are unlikely to give highly correlated decryption error terms. Secondly, we devise a degraded channel which, different from the independent assumption in existing literature, can be deemed as a worse channel than what the decryption error is distributed in real. Thirdly, we employ polar codes for error correction. It gives competitive decoding capability thanks to the polarization phenomenon and soft-decision decoding.

4 Description of the Cryptosystems

4.1 Public Key Encryption

The IND-CPA secure **public key encryption** scheme PolarLAC.PKE.CPA lays the foundation of the entire PolarLAC. It comprises three algorithms:

- The key generation algorithm KG, as illustrated in Algorithm 8.
- The encryption algorithm Enc, as illustrated in Algorithm 9.
- The decryption algorithm Dec, as illustrated in Algorithm 10.

PolarLAC.PKE.CPA is parameterized by the integers q, n, k, l_m, d , and the ternary distribution $\Psi_{1/4}$, $\Psi_{3/8}$ and $\Psi_{11/64}$. Using the notations defined in section 2, we give the description of key generation, encryption, decryption and their subroutines as follows.

Subroutines. Denote by $\mathcal{M}$ the message space $\{0, 1\}^{l_m}$. Depending the parameter settings of PolarLAC, we design a polar code $\text{Polar}[l_v, \mathcal{I}]$ which is uniquely determined by the code length l_v and information set $\mathcal{I} \subset \{0, 1, \dots, l_v - 1\}$. The subroutine **PolarEnc** takes the plaintext message $\mathbf{m}$ as input, assigns it to the entries of the vector $\mathbf{u}$ associated with the information set $\mathcal{I}$, and yields the codeword $\hat{\mathbf{m}}$. After decryption, we derive a noisy version $\mathbf{y}$ of the codeword $\hat{\mathbf{m}}$. The **PolarDec** takes the vector $\mathbf{y} \in \mathbb{R}^{l_v}$ as an input and calculates the likelihood ration $\text{LR}_i = \frac{P(\mathbf{y}_i | \hat{\mathbf{m}}_i = 0)}{P(\mathbf{y}_i | \hat{\mathbf{m}}_i = 1)}$ of $\mathbf{y}_i$ with respect to the distribution of decryption error. In this work, we find a virtual worse version of the decryption error based on which the conditional probabilities and the likelihood ratios are calculated. The likelihood ratios are passed to the successive cancellation decoder [6,35] **SCDecoder**(y) which outputs $\mathbf{m} = \mathbf{u}_{\mathcal{I}}$.

Algorithm 1 PolarEnc($\mathbf{m} \in \mathcal{M}$)

Ensure: An encoding $\hat{\mathbf{m}} \in \{0, 1\}^{l_v}$.
1: $\mathbf{u} = \mathbf{0}$
2: $\mathbf{u}_{\mathcal{I}} \leftarrow \mathbf{m}$ for $\mathcal{I} \subset \{0, 1, \dots, l_v - 1\}$
3: $\hat{\mathbf{m}} \leftarrow \mathbf{u}^t \mathbf{G}$ // $\mathbf{G}$ is the generator matrix
4: **return** $\hat{\mathbf{m}}$

Algorithm 2 PolarDec($\mathbf{y} \in \{0, 1\}^{l_v}$)

Ensure: A bit string $\mathbf{m}$.

- 1: $\text{LR}_i = \frac{P(\mathbf{y}_i | \mathbf{m}_i=0)}{P(\mathbf{y}_i | \mathbf{m}_i=1)}$ for $i = 0, 1, \dots, l_v - 1$
 - 2: $\mathbf{u} \leftarrow \text{SCDecoder}(\text{LR})$
 - 3: $\mathbf{m} \leftarrow \mathbf{u}_{\mathcal{I}}$
 - 4: **return** $\mathbf{m}$
-

The subroutines $\text{Compr}_d(\cdot), \text{DeCompr}_d(\cdot)$ convert between elements of $\mathbb{Z}_q$ and $\mathbb{Z}_{2^d}$ for an integer d , where $1 < 2^d < q$. To be specific, given $x \in \mathbb{Z}_q$,

$$\text{Compr}_d(x) = \lceil \frac{x \cdot 2^d}{q} \rceil \mod 2^d,$$

and given $y \in \mathbb{Z}_{2^d}$, it gives that

$$\text{DeCompr}_d(y) = \lceil y \cdot \frac{q}{2^d} \rceil.$$

For $\mathbf{x} \in \mathbb{Z}_q^n$, $\text{Compr}_d(\mathbf{x})$ (resp. $\text{DeCompr}_d(\mathbf{x})$) means that Compr_d (resp. DeCompr_d) is applied to $\mathbf{x}$ component-wise. The subroutines $\text{Compr}_d, \text{DeCompr}_d$ are described in Algorithm 3 and 4, respectively.

Algorithm 3 Compr_d($x \in \mathbb{Z}_q$)

Ensure: An integer $y \in \mathbb{Z}_{2^d}$.

- 1: $y \leftarrow \lceil \frac{2^d \cdot x}{q} \rceil \mod 2^d$
 - 2: **return** y
-

Algorithm 4 DeCompr_d($x \in \mathbb{Z}_{2^d}$)

Ensure: An integer $y \in \mathbb{Z}_q$.

- 1: $y \leftarrow \lceil \frac{q \cdot x}{2^d} \rceil$
 - 2: **return** y
-

As in Definition 5, the MLWE_{Rej} problem is defined by first drawing a polynomial number of standard MLWE samples $(\mathbf{a}, b) \in R_q^k \times R_q$, followed by a rejection sampling procedure. The i -th entry a_i of $\mathbf{a}$ is rejected if $\text{NTT}_j(a_i) = q - 1$ for any index $j \in \{0, \dots, n - 1\}$. The sampling procedure for $\mathbf{a}$ is detailed in Algorithm 5.

Algorithm 5 RejSampU($R_q; \text{seed}$)

Ensure: A sample $a \in R_q$

- 1: **if** $q = 257$ **then**
 - 2: **repeat**
 - 3: $\text{NTT}(a) \leftarrow \text{Samp}(U(\mathbb{Z}_q^n); \text{seed})$
 - 4: **until** $\|\mathbf{a}\|_\infty < q - 1$
 - 5: **else**
 - 6: $\text{NTT}(a) \leftarrow \text{Samp}(U(\mathbb{Z}_q^n); \text{seed})$
 - 7: **end if**
 - 8: **return** a
-

Observe that for identical input parameters, Algorithm 5 produces outputs following the same distribution as Algorithm 6, yet the latter avoids explicit rejection sampling. Consequently, we invoke the following algorithm to sample $A \in R_q^{k \times k}$ during the public-key generation phase in practice.

Algorithm 6 $\text{RejSampU}(R_q; \text{seed})$

Ensure: A sample $a \in R_q$
1: **if** $q = 257$ **then**
2: $\text{NTT}(a) \leftarrow \text{Samp}(U(\mathbb{Z}_{q-1}^n); \text{seed})$
3: **else**
4: $\text{NTT}(a) \leftarrow \text{Samp}(U(\mathbb{Z}_q^n); \text{seed})$
5: **end if**
6: **return** $a \in R_q$

A unique design feature of PolarLAC is the spectral rejection sampling filter for noise and secret polynomials, applied during key generation and encryption; further details are provided in Section 6. The final decryption noise consists of four i.i.d. terms, each being a product of two independent polynomials; these multiplicative terms induce correlation across the coefficients of the resulting noise vector. We denote one such term as $e \cdot r$, and the remaining three terms are handled identically.

We draw $e \leftarrow \Psi_\sigma$ and $r \leftarrow \Psi_\sigma$, meaning all coefficients of e and r are sampled i.i.d. from Ψ_σ . The spectral rejection sampling ensures that e and r satisfy $\|\text{FFT}(e)\|_\infty \leq T$ and $\|\text{FFT}(r)\|_\infty \leq T$. We found that rejecting a fraction of λ^{-3} extreme realizations suffices to construct suitable polar codes with sufficiently low decoding failure probability.

In the practical implementation of our algorithm, we adopt a tighter rejection threshold $T_0 < T$, which yields a rejection rate of approximately 1%. This choice effectively further reduces both the correlations. As a result, potential CCA-style attacks become infeasible. The subroutine $\text{RejSamp}(T, \Psi_\sigma; \text{seed})$ repeatedly samples a vector $\mathbf{x} \in R_q^k$ from Ψ_σ until the spectral norm satisfies that $\|\text{FFT}(x_i)\|_\infty \leq T$, as outlined in Algorithm 7.

Algorithm 7 $\text{RejSamp}(T, \Psi_\sigma; \text{seed})$

Ensure: A sample $\mathbf{x}$ such that $\|\text{FFT}(x_i)\|_\infty < T$ for $i = 1, \dots, k$
1: **for** $i = 1$ **to** k **do**
2: $\text{nonce} = 0$
3: **repeat**
4: $x_i \leftarrow \text{Samp}(\Psi_\sigma; \text{seed}, \text{nonce}++) \in R_q$
5: $\tilde{x}_i \leftarrow \text{FFT}(x_i) \in \mathbb{C}^n$
6: **until** $\|\tilde{x}_i\|_\infty \leq T$
7: **end for**
8: $\mathbf{x} = (x_i)_i$
9: **return** $\mathbf{x} \in R_q^k$

The algorithms. The algorithm PolarLAC.PKE.CPA.KG randomly generates a pair of public key and secret key (pk, sk) . As explained in Section 3.3, we apply rejection sampling to the public information $\mathbf{A}$ and $\mathbf{b}$ whose underlying hardness assumption is the module-LWE_{rej}. How to set the threshold γ for $\mathbf{b}$ is described in section 5.3.

Algorithm 8 PolarLAC.PKE.CPA.KG()

Ensure: A pair of public key and secret key (pk, sk) .

```

1: repeat
2:    $seed_a, seed \xleftarrow{\$} \mathcal{S}$ 
3:    $A \leftarrow \text{RejSampU}(R_q; seed_a) \in R_q^{k \times k}$ 
4:    $s, e \leftarrow \text{RejSamp}(T, \Psi_\sigma; seed) \in R_q^{k \times 1}$ 
5:    $b \leftarrow As + e \in R_q^{k \times 1}$ 
6:   if  $q = 257$  then
7:      $cnt = \text{count of 0 and 256 entries in } b$ 
8:   else
9:      $cnt = 0$ 
10:  end if
11: until  $cnt \leq \gamma$ 
12: return  $(pk := (seed_a, b), sk := s)$ 

```

The algorithm PolarLAC.PKE.CPA.Enc on input pk and a message m , encrypts m with the randomness seed. In case that no seed is provided, the process will be randomized. Otherwise, the encryption is deterministic for the same seed.

Algorithm 9 PolarLAC.PKE.CPA.Enc($pk = (seed_a, b), m \in \mathcal{M}; seed \in \mathcal{S}$)

Ensure: A ciphertext c .

```

1:  $A \leftarrow \text{RejSampU}(R_q; seed_a) \in R_q^{k \times k}$ 
2:  $\hat{m} \leftarrow \text{PolarEnc}(m) \in \{0, 1\}^n$ 
3:  $r, e_1 \leftarrow \text{RejSamp}(T, \Psi_\sigma; seed) \in R_q^{k \times 1}$ 
4:  $e_2 \leftarrow \text{Samp}(\Psi_\sigma; seed) \in R_q$ 
5:  $c_1 \leftarrow A^t r + e_1 \in R_q^{k \times 1}$ 
6:  $c'_2 \leftarrow b^t r + e_2 + \bar{q} \cdot \hat{m} \in R_q$ 
7:  $c_2 \leftarrow \text{Compr}_d(c'_2) \in \mathbb{Z}_{2d}^n$ 
8: return  $c := (c_1, c_2) \in R_q^{k \times 1} \times \mathbb{Z}_{2d}^n$ 

```

The algorithm PolarLAC.PKE.CPA.Dec on input sk and a ciphertext c , recovers the corresponding message m .

Algorithm 10 PolarLAC.PKE.CPA.Dec($sk = s, c = (c_1, c_2)$)

Ensure: A plaintext m .

```

1:  $u \leftarrow s^t c_1 \in R_q$ 
2:  $c'_2 \leftarrow \text{DeCompr}_d(c_2) \in R_q$ 
3:  $c_m \leftarrow c'_2 - u \in R_q$ 
4:  $m \leftarrow \text{PolarDec}(c_m)$ 
5: return  $m$ 

```

4.2 Key Encapsulation Mechanism

The IND-CCA secure key encapsulation mechanism PolarLAC.KEM.CCA is obtained by applying the FO transform [19,24] to PolarLAC.PKE.CPA, which comprises the following three algorithms:

- The key generation algorithm KG, as illustrated in Algorithm 11;
- The encapsulation algorithm Enc, as illustrated in Algorithm 12;
- The decapsulation algorithm Dec, as illustrated in Algorithm 13.

Notations. The notations for the description of PolarLAC.KEM.CCA are the same with those of PolarLAC.PKE.CPA. Additionally, we use hash functions $G : \{0, 1\}^* \rightarrow \mathcal{S} \times \{0, 1\}^{l_k}$, and $H : \{0, 1\}^* \rightarrow \{0, 1\}^{l_k}$ for the implicit rejection FO transformation [20,24,32] and generating the encapsulated key, where l_k denotes the length of the session key, and will vary depending on different security levels. In PolarLAC.KEM.CCA, we always set $l_k = l_m$. The concrete choice of the hash functions G and H will be specified in Section 5.

The algorithm PolarLAC.KEM.CCA.KG invokes the PolarLAC.PKE.CPA.KG algorithm to generate pk, sk' , then it picks a random seed z . The secret key sk includes sk', z and the public key pk .

Algorithm 11 PolarLAC.KEM.CCA.KG()

Ensure: A pair of public key pk and secret key sk .

- 1: $(pk, sk') \leftarrow \text{PolarLAC.PKE.CPA.KG}$
 - 2: $z \xleftarrow{\$} \mathcal{S}$
 - 3: $sk := (sk', pk, z)$
 - 4: **return** (pk, sk)
-

The algorithm PolarLAC.KEM.CCA.Enc on input pk and a seed seed_m , generates a message m , and encrypts m by invoking PolarLAC.PKE.CPA.Enc with pk, m and the randomness seed , which is generated from a hash of m and pk .

Algorithm 12 PolarLAC.KEM.CCA.Enc($pk; \text{seed}_m$)

Ensure: A ciphertext and encapsulated key pair (c, K) .

- 1: $m \leftarrow \text{Samp}(U(\mathcal{M}); \text{seed}_m) \in \mathcal{M}$
 - 2: $(\text{seed}, K) \leftarrow G(m || pk)$
 - 3: $c \leftarrow \text{PolarLAC.PKE.CPA.Enc}(pk, m; \text{seed})$
 - 4: **return** (c, K)
-

The decapsulation algorithm PolarLAC.KEM.CCA.Dec on input sk and a ciphertext, recovers a message by invoking PolarLAC.PKE.CPA.Dec. Then it verifies the correctness of the decryption by a re-encryption process. In case that the verification is passed, it returns the encapsulated key. Otherwise, it generates a pseudorandom string from the secret key z and the ciphertext.

Algorithm 13 PolarLAC.KEM.CCA.Dec(sk, c)

Ensure: An encapsulated key K .

- 1: parse $sk \rightarrow (sk', pk, z)$
 - 2: $m \leftarrow \text{PolarLAC.PKE.CPA.Dec}(sk, c)$
 - 3: $(K, \text{seed}) \leftarrow G(m || pk)$
 - 4: $K' \leftarrow H(z || c)$
 - 5: $c' \leftarrow \text{PolarLAC.PKE.CPA.Enc}(pk, m; \text{seed})$
 - 6: **if** $c' \neq c$, **then**
 - 7: $K \leftarrow K'$
 - 8: **end if**
 - 9: **return** K
-

While the standard IND-CCA security considers only a single challenge public key and a single challenge ciphertext, it does not provide the same security guarantee in the multi-target scenario where the adversary aims to break at least one among many challenge ciphertexts. The FO transform may lead to more than expected security loss in this multi-target setting, and

attacks have been identified against FO-based KEMs (e.g., FrodoKEM-640 and HQC-128)[9]. An effective way to protect FO-based KEMs against this attack is to add a random but public salt during encapsulation [21]. So, we give the salted version of encapsulation and decapsulation in Algorithm 14 and 15, respectively.

Algorithm 14 Salted version of PolarLAC.KEM.CCA.Enc(pk ; seed $_m$, salt)

Ensure: A ciphertext and encapsulated key pair (c, K) .

- 1: $m \leftarrow \text{Samp}(U(\mathcal{M}); \text{seed}_m) \in \mathcal{M}$
- 2: $(\text{seed}, K) \leftarrow G(m || pk || \text{salt})$
- 3: $\hat{c} \leftarrow \text{PolarLAC.PKE.CPA.Enc}(pk, m; \text{seed})$
- 4: $c := (\text{salt}, \hat{c})$
- 5: **return** (c, K)

Algorithm 15 Salted version of PolarLAC.KEM.CCA.Dec(sk, c)

Ensure: An encapsulated key K .

- 1: parse $sk \rightarrow (sk', pk, z)$
- 2: parse $c \rightarrow (\text{salt}, \hat{c})$
- 3: $m \leftarrow \text{PolarLAC.PKE.CPA.Dec}(sk', \hat{c})$
- 4: $(K, \text{seed}) \leftarrow G(m || pk || \text{salt})$
- 5: $K' \leftarrow H(z || \hat{c})$
- 6: $\hat{c}' \leftarrow \text{PolarLAC.PKE.CPA.Enc}(pk, m; \text{seed})$
- 7: **if** $\hat{c}' \neq \hat{c}$, **then**
- 8: $K \leftarrow K'$
- 9: **end if**
- 10: **return** K

5 Parameter Choices

5.1 Modulus

The modulus is selected by jointly considering algebraic structure, correctness margin, and implementation efficiency. The detailed reasons are discussed as follows.

For Security Level of 128, 256. For security 128 and 256, we mainly compared three candidate values, namely $q = 251$, $q = 256$, and $q = 257$. All three are close to 2^8 , and thus belong to the regime of small moduli near the 8-bit boundary. From an implementation perspective, they all incur relatively low bit-width overhead in data representation. However, they differ substantially in terms of polynomial multiplication efficiency and the overall system cost. For $q = 251$, the main advantage is that it fits naturally into a single byte, leading to compact storage. Its drawback, however, is the lack of the required NTT-friendly algebraic structure, so it cannot directly support an efficient NTT at the target transform length. Under this modulus, polynomial multiplication typically has to rely on lifting to an auxiliary NTT-friendly modulus with a larger bit-width. As a result, the internal arithmetic of the NTT still has to be performed over a modulus of at least 14 bits with our dimension set, which prevents the implementation from fully exploiting the advantages of a genuinely small modulus in modular reduction and butterfly operations.

For $q = 256$, the advantage is even more pronounced than for $q = 251$, primarily due to its power-of-two structure. Under modulus 256, data representation, truncation, reduction, and storage aligned with byte boundaries are all particularly natural. Such operations can be efficiently mapped to bitwise manipulations or machine-word operations, which provides clear engineering benefits at the implementation level. In particular, moduli of the form 2^k exhibit

the most regular structure for representation and low-level modular arithmetic. Nevertheless, 256 is also an NTT-unfriendly modulus, which means that the core polynomial multiplication still remains a bottleneck. This implies that its advantages at the representation level cannot be directly translated into efficiency gains for the core NTT operators.

By contrast, although $q = 257$ exceeds the natural one-byte representation range and therefore typically requires 16-bit storage for coefficients, it can directly support NTT operations. Since the Fermat prime $257 = 2^8 + 1$ has a favorable algebraic structure, the scheme can utilize lightweight NTT, pointwise multiplication, and inverse NTT directly modulo 257, while using low-cost conditional reductions in the butterfly operations. This substantially reduces the implementation complexity of polynomial multiplication. Moreover, the polynomial $\mathbf{a}$ can be sampled directly in the NTT domain, thereby avoiding an additional forward NTT and further reducing the computational overhead.

In summary, the main advantages of 251 and 256 lie in storage representation or the convenience of low-level modular arithmetic, but neither can directly support an efficient NTT. By contrast, although 257 requires a larger storage width, it provides significantly better implementation efficiency in the core polynomial arithmetic. Furthermore, the storage overhead introduced by 257 can be effectively mitigated through compression and compact encoding. Therefore, after jointly considering computational efficiency, implementation complexity, and bandwidth overhead, we ultimately selected $q = 257$ as the modulus in our scheme.

For Security Level of 512. We provide two parameter sets, both of which achieve the target security level. The larger modulus $q = 769$ also supports the partially expanded NTT structure. Moreover, for the scenario with unlimited query numbers, the modulus $q = 769$ ensures a DFR below 2^{-512} . In contrast, when the number of queries is limited to 2^{80} , the DFR does not need to meet the stringent requirement of the 512-bit security level. In this case, the alternative parameter set with $q = 257$ is recommended, which achieves a DFR of 2^{-323} .

5.2 Distributions

Noise and Secrets. There are three criteria for the choice of the distribution for the error and secret terms for MLWE. Firstly, entropy of the errors and the secrets must be large enough to guarantee the hardness of the module-LWE problem. Secondly, the errors and the secrets must be small enough to guarantee the correctness of the decryption algorithm. Thirdly, they can be efficiently implemented.

In the literature, the security of lattice-based schemes is fundamentally grounded in worst-to-average-case reductions that mandate a discrete Gaussian distribution [33]. However, in practical instantiations the centered binomial distribution and ternary distribution are preferred as implementation-friendly alternatives [5,10,36].

In PolarLAC, we adopt the following ternary distributions after carefully balancing decryption correctness, cryptographic security strength, and implementation efficiency. The efficient realization of sampling will be given in Section 8.

1. $\Psi_{\frac{1}{4}}: \Pr[x = 0] = 3/4, \Pr[x = \pm 1] = 1/8.$
2. $\Psi_{\frac{3}{8}}: \Pr[x = 0] = 5/8, \Pr[x = \pm 1] = 3/16.$
3. $\Psi_{\frac{11}{64}}: \Pr[x = 0] = 53/64, \Pr[x = \pm 1] = 11/128.$

The Public Key. For the modulus $q = 257$. The public matrix is sampled from the uniform distribution over R_q . For the parameter sets with $q = 257$, this sampling can be interpreted as rejection sampling over $\mathbb{Z}_{257}$: each candidate coefficient is first sampled uniformly from $\mathbb{Z}_{257}$, and the value 256 is rejected. Equivalently, conditioned on acceptance, each coefficient is uniformly distributed over $\{0, \dots, 255\}$, which is precisely the coefficient range used in the implementation. At the coefficient level, the rejection probability is

$$p_{\text{rej}}^{(1)} = \frac{1}{257}.$$

For a polynomial of degree $n - 1$, the probability that at least one coefficient is rejected is

$$p_{\text{rej}}^{(n)} = 1 - \left(1 - \frac{1}{257}\right)^n = 1 - \left(\frac{256}{257}\right)^n.$$

This rejection only affects the generation of the public uniformly random matrix and does not introduce additional error terms into either encryption or decryption. After rejection sampling, the resulting public matrix remains uniformly distributed over the accepted coefficient range and remains independent of the secret and error distributions. Consequently, the small error terms that determine the decryption-failure probability remain unchanged. Therefore, the rejection event is a sampling-level event rather than a decryption-failure event. Its influence on the final decryption error rate is negligible and is not included as an additional noise contribution in the correctness analysis.

Meanwhile, its rejection rate is approximately $1 - e^{-O(1)} < 1 - 1/\text{poly}(n)$, which guarantees the hardness of the corresponding MLWE_{Rej} .

For the modulus $q = 769$. For PolarLAC-512*, the public matrix is sampled over $\mathbb{Z}_{769}$. Since 769 is not a power of two and differs substantially from primes such as 257, direct byte-wise sampling cannot generate unbiased coefficients. We thus adopt grouped rejection sampling in our sampling routine. Concretely, we generate a 48-bit candidate integer to produce five coefficients. If the candidate value is less than 769^5 , we accept it and decompose it in base 769 to retrieve five coefficients lying in $\mathbb{Z}_{769}$; otherwise, the candidate is discarded and we resample. For the remaining four coefficients, we follow an identical procedure with a 39-bit candidate and acceptance threshold 769^4 . The rejection probabilities for the two groups are given by

$$p_{\text{rej}}^{(5)} = 1 - \frac{769^5}{2^{48}}, \quad p_{\text{rej}}^{(4)} = 1 - \frac{769^4}{2^{39}}.$$

Upon acceptance, each base-769 digit follows the uniform distribution over $\mathbb{Z}_{769}$, and the resulting public matrix is independent of both the secret and error distributions. Hence, the rejection event only impacts the runtime of public matrix generation; it introduces no extra noise terms during encryption or decryption, and imposes no negative impact on the overall security.

5.3 Data Compression

Compression of the Public Key. The public key contains the public seed and the vector component generated over R_q . To reduce the public key size while preserving exact decodability, we use two lossless compression methods according to the modulus.

Fixed zero selector for $q = 257$. For the parameter sets with $q = 257$, each coefficient of the public key polynomial lies in $\mathbb{Z}_{257}$, which is very close to 8-bit size. A direct 9-bit representation would increase the public key size, while storing only the lower 8 bits makes the two values 0 and 256 indistinguishable. Therefore, we use a fixed zero-selector method. For each coefficient $a_i \in \mathbb{Z}_{257}$, we store

$$\bar{a}_i = a_i \bmod 256$$

as one byte. The only ambiguity occurs when $\bar{a}_i = 0$, where the original coefficient is either 0 or 256. A selector bit is therefore used to distinguish these two cases. If the number of such low-zero positions exceeds the fixed selector capacity, the public key generation is rejected and rerun.

Let τ be the number of selector bytes appended to each polynomial. The selector capacity is $\gamma = 8\tau$ bits. Although the compressed public key component $\mathbf{b}$ is not sampled directly from the uniform distribution. Under this model, the number of low-zero positions in one polynomial satisfies

$$X \sim \text{Bin}\left(n, \frac{2}{257}\right),$$

because both 0 and 256 are mapped to the same low byte 0.

Thus, the rejection probability for one polynomial is

$$p_{\text{rej}}^{\text{poly}}(n, \tau) = \Pr[X > 8\tau] = 1 - \sum_{i=0}^{8\tau} \binom{n}{i} \left(\frac{2}{257}\right)^i \left(\frac{255}{257}\right)^{n-i}.$$

For a public-key vector of rank k , the public-key rejection probability is

$$p_{\text{rej}}^{\text{pk}}(n, k, \tau) = 1 - \left(1 - p_{\text{rej}}^{\text{poly}}(n, \tau)\right)^k.$$

This rejection only affects key generation and does not introduce any lossy compression error into the public key.

The current selector parameters are summarized in Table 1. Owing to our adoption of the module structure, the total tail size of the full public key vector amounts to $k\tau$ bytes. It can be observed that the rejection rate associated with b is extremely small under our parameter setup. Combined with the rejection rate of a discussed earlier, the overall composite rejection rate can be bounded by $1/O(1)$. This suffices to guarantee that the MLWE_{Rej} problem is still hard.

Parameter set	n	k	Selector bytes per polynomial τ	$p_{\text{rej}}^{\text{pk}}(n, k, \tau)$
PolarLAC-Light	256	2	1 byte	$4.23 \times 10^{-4} \approx 2^{-11.21}$
PolarLAC-128	256	2	1 byte	$4.23 \times 10^{-4} \approx 2^{-11.21}$
PolarLAC-256	512	2	2 bytes	$1.84 \times 10^{-6} \approx 2^{-19.05}$
PolarLAC-512*	1024	2	2 bytes	$6.88 \times 10^{-3} \approx 2^{-7.18}$

Table 1: Fixed zero-selector parameters for public-key compression

CRT-style packing for $q = 769$. For PolarLAC-512, the modulus is $q = 769$. Since a byte-oriented representation is no longer sufficient, the public-key polynomial is compressed by a lossless CRT-style base- q packing method. The coefficients are first grouped in blocks of five:

$$A = a_0 + a_1q + a_2q^2 + a_3q^3 + a_4q^4, \quad 0 \leq a_i < q.$$

Since $q = 769$ and $769^5 < 2^{48}$, each group of five coefficients can be represented by 48 bits. For $n = 1024$, the first 1020 coefficients form 204 such groups, and the remaining four coefficients are packed into a 39-bit integer because $769^4 < 2^{39}$. Therefore, the size of one compressed polynomial is

$$\left\lceil \frac{204 \cdot 48 + 39}{8} \right\rceil = 1229 \text{ bytes.}$$

This encoding is injective over $\mathbb{Z}_{769}^{1024}$, so decompression exactly recovers the original public-key polynomial. In the current implementation, the same lossless representation is also used for the NTT-domain vector component that appears in the PolarLAC-512 public key and ciphertext.

Compression of the Ciphertext. The variance of the compression error with respect to Algorithm 3 and 4 is $\frac{q^2}{12 \times 4^d}$ if we represent a number modulo q using d bits.

For the Fermat prime $q = 257$, we represent a number modulo 257 with exactly 1 byte and with a small compression error by mapping 256 to 0, which creates only an off-by-one difference in modulo q arithmetic. The variance of the compression error is $1/q$. If the ciphertext c_1 for $q = 257$ is represented using 8 bits, it will induce an additional decryption error term $\Delta_{c_1} \cdot s$ with a variance of $n \cdot k \cdot \sigma^2/q$.

5.4 Spectral Rejection Sampling

PolarLAC employs a spectral rejection sampling technique to handle the error correlations. There are CCA attacks that increase the decryption failure rate by leveraging special patterns of the ternary samples. Some effective patterns are defined in [22]. To understand how attacks of this kind work may be nontrivial in time domain, whereas, their behavior in frequency domain is more straightforward. Before giving the theoretical interpretation, we firstly show some examples in [22] which devised attacks of this kind against LAC. For LACv2-256, a routinely sampled polynomial $e_1(x)$ will have a power spectrum in frequency domain as in Fig. 3a. In [22], a pattern called Type-1 is identified if a polynomial $e_1(x)$ comprises consecutive l_1 coefficients equal to +1 (or -1). It is derived in [22] that it takes a computational cost of 2^{120} to find a Type-1 $e_1(x)$ with $l_1 = 65$. The power spectrum in frequency domain for such a $e_1(x)$ is depicted in Fig. 3b. A dedicated $s(x)$ is also devised in accordance with the Type-1 $e_1(x)$. It takes a computational cost of 2^{64} to meet such an $s(x)$ with a power spectrum as in Fig. 3c. It is observed that they have strong zero-frequency signal simultaneously.

The TYPE-2b pattern is devised for the fixed hamming weight design. It refers to a polynomial e_1 containing a contiguous interval of length $l_1 + l_0$, starting at a specific offset. Within this interval, even (resp. odd) indices contain as $l_1/2$ coefficients equal to +1 (resp. -1) and $l_0/2$ coefficients equal to 0. We plot the power spectrum of such e_1 with $l_1 = 114, l_0 = 80$ starting after position 801 in Fig. 3d. This amounts to a computation cost of 2^{120} . The other dedicated $s(x)$ is also devised for Type-2b $e_1(x)$ both of which has strong strong power at high frequency (i.e. 1/2 of the Nyquist frequency) domain.

The polynomial product $e_1(x)s(x)$ is a part of the decryption error for LAC. To illustrate how the dedicated patterns deteriorate the DFR, we define $\mathbf{z} = \mathbf{E}_1 \cdot \mathbf{s}$, where $\mathbf{E}_1$ is the negacyclic matrix derived by polynomial $e_1(x)$ and $\mathbf{s}$ is a vector defined by coefficients of $s(x)$. By applying FFT decomposition to $\mathbf{E}_1$, we will have

$$\begin{aligned} \mathbf{z} &= \mathbf{E}_1 \cdot \mathbf{s} \\ &= U^H \cdot \text{diag}(\sigma_i(e_1))_i \cdot U \cdot \mathbf{s} \\ &= 1/\sqrt{n} \cdot U^H \cdot (\sigma_i(e_1) \odot \sigma_i(s))_i, \end{aligned}$$

where U is the unitary matrix corresponding to FFT decomposition of a negacyclic matrix, $\sigma_i(x)$ is the FFT of x , $\text{diag}(\cdot)_i$ represents a diagonal matrix, H indicate the Hermite transpose and $\odot$ represents component-wise product. So, if $\sigma_i(e_1)$ and $\sigma_i(s)$ has a big magnitude at the same frequency point, $\sigma_i(e_1) \odot \sigma_i(s)$ will have a big magnitude as well. After the rotation operation of U^H , this high power component will make multiple coefficients of $\mathbf{z}$ big simultaneously. This symptom amounts to the so-called “correlation” or “dependency” in the context of ring/module-LWE in the literature.

Given the rationale of “correlation”, a straightforward method to overcome the attacks of this kind leveraging the “correlation” is a rejection sampling filter in frequency domain such that the abnormal $e_1(x)$ and $s(x)$ with far stronger than expected components at certain frequencies will be removed. We in this submission setup a threshold for the spectral norm T_0 of e_1 such that e_1 is used only if $\max |\sigma_i(e_1)| \leq T_0$. Because the multiplication is commutative over the quotient ring $\mathbb{Z}[x]/(1+x^n)$, the same protection also applies to s . A step-by-step procedure of spectral rejection sampling is in Algorithm 7.

Our analysis reveals that a threshold T yielding a rejection rate below 10^{-6} , combined with channel degradation and polar coding techniques, is sufficient to defend against targeted attacks of this type. In Table 2, T_0 is the threshold we used in implementation, while T is the one we used to construct a degraded channel and corresponding polar codes. See the construction details in Section 6. Channel degradation introduces a virtual decryption error corresponding to a worse signal-to-noise ratio (SNR) than the actual decryption error encountered in practice. To err on the side of caution, we choose a T_0 that achieves an rejection rate of approximately 1% per polynomial sample for each security category, see Table 2. Furthermore, we verified the correctness of our algorithms under a more generalized attack devised in frequency domain. The generalized attack can be viewed as hybrid of components of different frequencies. The detailed analysis and theories are given in Section 6.

category	T_0	rejection rate	T	rejection rate
PolarLAC-Light	24	1.7 %	36	$< 128^{-3}$
PolarLAC-128	30	1.1%	42	$< 128^{-3}$
PolarLAC-256	36	1.1%	58	$< 256^{-3}$
PolarLAC-512	43	1.4%	80	$< 512^{-3}$
PolarLAC-512*	64	1.2%	116	$< 512^{-3}$

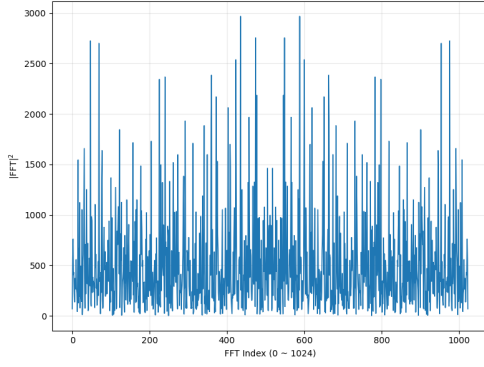
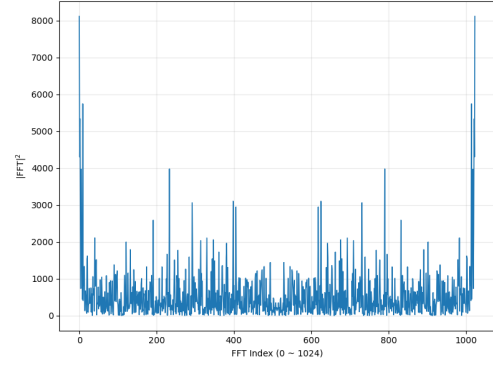
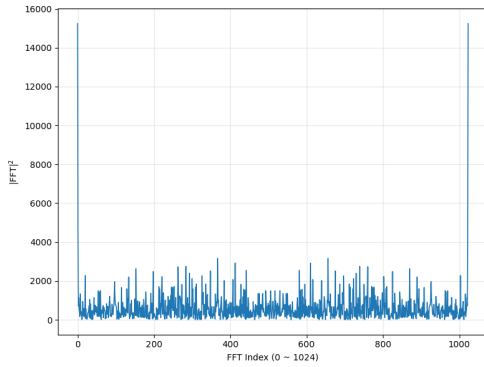
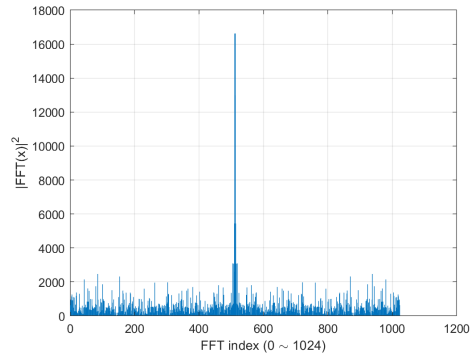
Table 2: Threshold values of T_0, T and rejection rates.(a) average-case $e_1(x)$,(b) Type-1 $e_1(x)$, $\log_2(\text{Cpre}) = 120$ (c) A dedicated $s(x)$ for Type-1 $e_1(x)$ (d) Type-2b $e_1(x)$, $\log_2(\text{Cpre}) = 120$

Fig. 3: Frequency spectrum: average-case vs. under cca attack

5.5 Recommended Parameter Categories

We give parameter sets according to different security categories and security notions. The IND-CCA version of PolarLAC is built upon PolarLAC.PKE.CPA, and its parameter sets are specified in Table 3. For scenarios where the security estimation by refined BKZ suffices and the lightweightness is more concerned, the PolarLAC-light is recommended. In unrestricted query count scenarios, PolarLAC-128, PolarLAC-256, and PolarLAC-512* are recommended, as their decryption failure rates (DFR) match the corresponding security parameters. By contrast, for settings with bounded query counts, PolarLAC-512 is preferable to PolarLAC-512*; it delivers better computational efficiency, smaller ciphertext/key size. Its DFR is less favorable relative to PolarLAC-512*, yet remains competitive overall.

Categories	k	n	q	dis	R	l_v	d_1	d_2	pk	sk	ct	DFR	Bit-Security (C/Q)	
													core-SVP	refined BKZ
PolarLAC-Light	2	256	257	$\Psi_{1/4}$	1/2	256	8	3	530	1570	608	2^{-159}	121.6/ 111.7	143.8/ 138.8
PolarLAC-128	2	256	257	$\Psi_{3/8}$	1/2	256	8	4	530	1570	640	2^{-146}	132.5/ 122.6	154.5/ 148.1
PolarLAC-256	2	512	257	$\Psi_{1/4}$	1/2	512	8	4	1060	3140	1280	2^{-265}	261.6/ 240.8	280.8/ 263.4
PolarLAC-512	2	1024	257	$\Psi_{11/64}$	1/2	1024	8	4	2116	6276	2560	2^{-324}	512.1/ 482.5	527.3 / 499.3
PolarLAC-512*	2	1024	769	$\Psi_{3/8}$	1/2	1024	9.60 [†]	4	2522	6682	2970	2^{-545}	527.1/ 484.7	540.1/ 500.3

dis secret and noise distributions

l_v code length of polar codes

d_1, d_2 bits per coefficient of $\mathbf{c}_1, \mathbf{c}_2$

sk secret key size (bytes)

pk public key size (bytes)

ct ciphertext size (bytes)

DFR decryption failure rate

R message length/code length

[†] The compact data storage for $\mathbb{Z}_{769}$ takes 9.6 bits per number on average

Table 3: Parameter Sets of PolarLAC.KEM.CCA

6 Decryption Failure Rate Analysis

We define an abstract channel model for the decryption procedure of PolarLAC as

$$c_2 - \mathbf{c}_1 \cdot \mathbf{s} = \lfloor q/2 \rfloor \cdot \text{PolarEnc}(\mathbf{m}) + \mathbf{e}^t \cdot \mathbf{r} - \mathbf{s}^t \cdot \mathbf{e}_1 + e_2 + n_{c_1} + n_{c_2} \bmod q,$$

where n_{c_2} denotes the i.i.d. compression noise associated with ciphertext c_2 , and n_{c_1} corresponds to the compression noise of $\mathbf{c}_1$. In the sequel, we denote by h the overall noise

$$h = \mathbf{e}^t \cdot \mathbf{r} - \mathbf{s}^t \cdot \mathbf{e}_1 + e_2 + n_{c_1} + n_{c_2}.$$

It is convincing that the correlation between distinct coefficients of n_{c_1} is negligible: compression noise only appears at positions where coefficients of $\mathbf{c}_1$ equal 256, leaving all other entries noise-free. When estimating the decryption failure rates, we viewed $e_2 + n_{c_1} + n_{c_2}$ as an i.i.d. Gaussian distribution. Full details on the ciphertext compression scheme for $\mathbf{c}_1$ are provided in Section 5.3.

The dominant term of the decryption error is $\mathbf{e}^t \cdot \mathbf{r} - \mathbf{s}^t \cdot \mathbf{e}_1$, where the 4 summands follow the same distribution independently. To simplify notation in the sequel, we rewrite the expression as $h' = \mathbf{e}^t \cdot \mathbf{r} - \mathbf{s}^t \cdot \mathbf{e}_1 = e \cdot r + e' \cdot r' - s \cdot e_1 - s' \cdot e'_1$, where e, r and all other ring elements are sampled from the ternary distribution Ψ_σ . Under the CCA security model, it is risky to assume the terms of polynomial products to be i.i.d. An CCA attacker may select some adversarial r and e (either explicitly or implicitly), such that their the coefficient of $e \cdot r$ are highly correlated. It becomes complicated to give a reliable DFR estimation. To address this, we propose a suite of

techniques—including frequency-domain rejection sampling filtering, channel degradation, and polar codes with soft-decision decoding to guarantee robust and conservative correctness bounds for PolarLAC.

6.1 The Thorn Structures of Polynomial Convolution

The impact of correlated decryption errors has been extensively investigated in prior works [17,15,16,13,14,13]. Existing results reveal that correlations among coefficients of polynomial products lead to underestimated DFR bounds for schemes equipped with error-correcting codes. Nevertheless, there remains an open gap regarding two critical problems: how to rigorously characterize such correlation structures, and how to design error-correcting codes tailored to correlated noise distributions. We bridge this gap in PolarLAC.

Define e and r in $R = \mathbb{Z}[x]/(x^n + 1)$ where n is a two-power number. Without loss of generality, we study the correlation of $e \cdot r$. The conclusion applies to the other 3 summands, as well. We have extensively discussed the correlation introduced by the polynomial product $e \cdot r$ in [11]. Briefly speaking, in the frequency domain, the entries of $\text{FFT}(e \cdot r)$ are mutually independent, where each entry equals to the product of two independent identically distributed complex Gaussian variables.

Though $\text{FFT}(e \cdot r)$ has i.i.d. coordinates, its distribution is non-spherical (i.e. Fig. 4). As the ℓ_2 -norm grows, samples asymptotically cluster near the complex coordinate axes. The underlying number field we employ has a simple structure, such that the FFT and inverse FFT only perform scaling and rotation without distorting sample geometry. Consequently, the inverse FFT preserves this non-spherical thorn structure in the time domain, which induces correlation among the coefficients of $e \cdot r$.

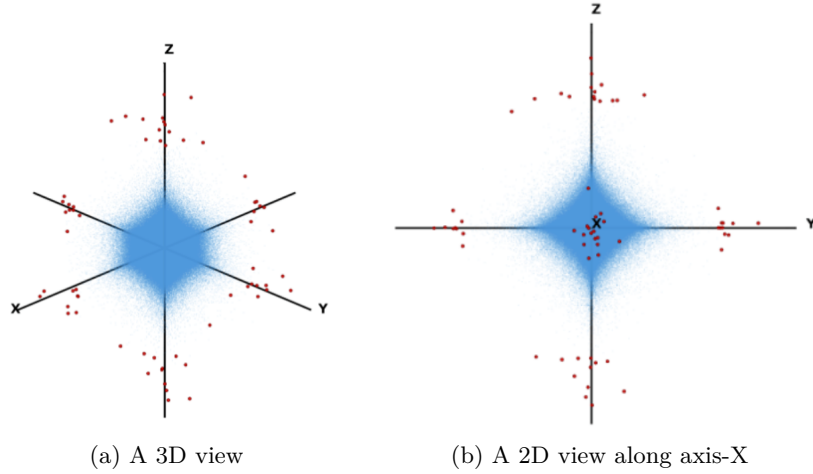


Fig. 4: The distribution of the magnitude of $\text{FFT}(e \cdot r)$.

6.2 A Degraded Channel Model

Although we have partially characterized the correlations among the coefficients of $e \cdot r$, constructing rigorous codes tailored to $e \cdot r$ requires deriving the full n -dimensional joint distribution over its coefficients — a task computationally infeasible under current computational capabilities. For this reason, we instead construct codes with respect to a virtual stronger noise. The qualifier “stronger” means we provide a heuristic rule for selecting this virtual noise and empirically confirm that the virtual noise always exhibits larger magnitudes than the true noise across all statistically relevant regimes. Our selection rule avoids the naive independence assumption; we have formally proven in [11] that the independence assumption always underestimates the

magnitude of the true noise. We in this section give a suite of methodologies to construct the virtual and worse channel, manage the correlation, and design polar codes.

We construct a virtual noise vector $\mathbf{e}_{\text{vir}}$ that is memoryless, symmetric, and of elevated noise magnitude. This virtual noise is obtained via a two-step procedure. The raw noise $\mathbf{e} \cdot \mathbf{r}$ suffers two unfavorable properties: its coefficients are correlated (hence non-memoryless), and its distribution lacks full rotational symmetry (the thorn-shaped mass concentrates along specific spatial directions). In Step 1, we build an intermediate noise $\mathbf{e}_{\text{deg}}$ from the raw noise to restore full distributional symmetry while increasing its entropy. In Step 2, we transform $\mathbf{e}_{\text{deg}}$ into the final virtual noise $\mathbf{e}_{\text{vir}}$. This $\mathbf{e}_{\text{vir}}$ satisfies three key properties: coordinate-wise independence (memorylessness), full symmetry, and larger noise magnitude.

Step 1: Noise Symmetrization via Orthogonal Rotation. Our first transformation step produces a symmetric intermediate channel as a prerequisite for constructing the final amplified virtual noise. Symmetry here means the distribution’s probability mass spreads uniformly over all spatial directions. The raw noise $\mathbf{e} \cdot \mathbf{r}$ does not follow a spherical distribution. We resolve this asymmetry by applying uniformly random orthogonal rotation matrices to the noise vector. Formally, let $\mathbf{e}_{\text{real}}$ denote the coefficient vector of the raw ring product $e_1 \cdot s$. We sample an orthogonal matrix O uniformly at random and define the intermediate rotated noise as $\mathbf{e}_{\text{deg}} = O \mathbf{e}_{\text{real}}$.

We claim that the random rotation increases the entropy of $\mathbf{e}_{\text{real}}$ for the following reasons. Firstly, since the condition does not increase the entropy, we have

$$h(\mathbf{e}_{\text{deg}}) \geq h(\mathbf{e}_{\text{deg}}|O).$$

The conditional entropy is derived by taking the expectation of $h(\mathbf{e}_{\text{deg}}|O = o)$ over all possible realization of O . Since O is uniformly sampled from a group of rotations, we have

$$h(\mathbf{e}_{\text{deg}}|O) = \mathbb{E}_O(h(\mathbf{e}_{\text{deg}}|O = o)) = h(\mathbf{e}_{\text{deg}}|O = o) = h(o \cdot \mathbf{e}_{\text{real}}).$$

Secondly, the entropy of $\mathbf{e}_{\text{real}}$ will be preserved after a rotation by any realization o . So, we have

$$h(o \cdot \mathbf{e}_{\text{real}}) = h(\mathbf{e}_{\text{real}}).$$

Combining the two statements yields $h(\mathbf{e}_{\text{deg}}) \geq h(\mathbf{e}_{\text{real}})$. Equality holds if and only if the mutual information between $\mathbf{e}_{\text{deg}}$ and O vanishes, i.e., $I(\mathbf{e}_{\text{deg}}; O) = h(\mathbf{e}_{\text{deg}}) - h(\mathbf{e}_{\text{deg}} | O) = 0$. This condition is satisfied exactly when $\mathbf{e}_{\text{real}}$ follows a spherical Gaussian distribution. In contrast, $\mathbf{e}_{\text{real}}$ follows the thorn-shaped distribution visualized in Fig. 4. From the above derivation, we conclude that $\mathbf{e}_{\text{deg}}$ carries higher entropy and consequently induces a smaller channel capacity compared with $\mathbf{e}_{\text{real}}$. A further vital property is that the orthogonal operator O endows $\mathbf{e}_{\text{deg}}$ with a fully spherical distribution.

Step 2: Elevating Noise Magnitude for Memorylessness. While $\mathbf{e}_{\text{deg}}$ achieves spherical symmetry and elevated entropy, its coefficients remain statistically correlated. No standard polar code construction exists to handle correlated spherical noise of this form. We therefore introduce an additional transformation to produce memoryless noise, at the cost of raising the overall noise magnitude.

By construction, $\mathbf{e}_{\text{deg}}$ is spherical: its directional component is uniformly distributed over the entire ambient space, and its statistical behavior is fully captured by its Euclidean magnitude. We construct the target virtual noise $\mathbf{e}_{\text{vir}}$ to be simultaneously spherical, memoryless, and of substantially larger magnitude—equivalently, a spherical Gaussian random vector with a much larger second moment. Note that $\mathbf{e}_{\text{deg}}$ has the same length as $\mathbf{e}_{\text{real}}$ because the rotation O is orthogonal. So, $\mathbf{e}_{\text{vir}}$ has bigger amplitude than $\mathbf{e}_{\text{real}}$, as well.

Recall that orthogonal matrices preserve ℓ_2 norms, so $\|\mathbf{e}_{\text{deg}}\|_2 = \|\mathbf{e}_{\text{real}}\|_2$. It immediately follows that $\mathbf{e}_{\text{vir}}$ exhibits a larger amplitude than the original raw noise $\mathbf{e}_{\text{real}}$. To summarize, we construct $\mathbf{e}_{\text{vir}}$ via a two-step transformation pipeline. Each stage degrades the underlying channel quality, yet this controlled degradation comes with a critical tradeoff: the resulting noise structure enables systematic polar code construction. We take PolarLAC-128 for example, the

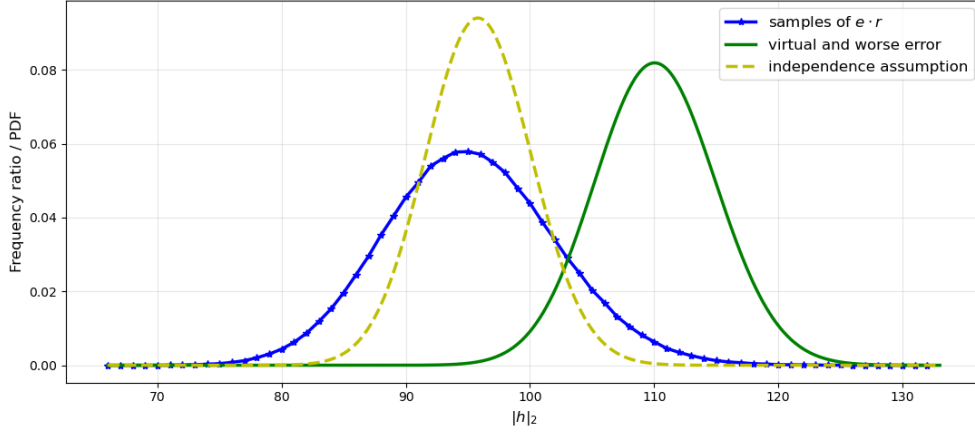


Fig. 5: P.D.F. as a function of l_2 -norm: $\mathbf{e}_{\text{real}}$ v.s. $\mathbf{e}_{\text{vir}}$ v.s. independence assumption.

$\mathbf{e}_{\text{vir}}$ we selected has a much bigger Euclidean magnitude than that of $\mathbf{e}_{\text{real}}$, see Fig. 5. Next, we will explain how to determine $\mathbf{e}_{\text{vir}}$.

Careful selection of the standard deviation for $\mathbf{e}_{\text{vir}}$ is essential. On one hand, if the standard deviation of $\mathbf{e}_{\text{vir}}$ is too small such that its typical magnitude does not exceed that of $\mathbf{e}_{\text{real}}$, one will overate the capability of error-correction codes. The code will perform far worse against the real noise $\mathbf{e}_{\text{real}}$ than expected. Estimating $\mathbf{e}_{\text{real}}$ via the naive independence assumption falls into this trap (see Fig. 5), as we have proven that the independence assumption always underestimates the true noise magnitude. On the other hand, an excessively large standard deviation for $\mathbf{e}_{\text{vir}}$ is also infeasible. One is likely to get an overestimated DFR and fail to upper bound the correctness on the real noise $\mathbf{e}_{\text{real}}$, though the codes may performs better on $\mathbf{e}_{\text{real}}$ than on $\mathbf{e}_{\text{vir}}$.

We select a proper standard deviation for the virtual Gaussian noise $\mathbf{e}_{\text{vir}}$ based on three critical observations. First, samples drawn from high-dimensional i.i.d. Gaussian distributions concentrate tightly around a fixed hypersphere. Second, one can easily verify that our zero-mean i.i.d. Gaussian noise defined over the ring's time domain remains i.i.d. complex Gaussian (excluding conjugate components) after transformation to the frequency domain. Third, as discussed in [11], the vast majority of $\text{FFT}(\mathbf{e}_{\text{real}})$ samples lie inside an ℓ_1 -norm contour; except for extreme outliers, the ℓ_1 norm of most $\text{FFT}(\mathbf{e}_{\text{real}})$ samples is bounded with overwhelming probability. Figure 4 visualizes this analogous distribution over real space, and the complex case exhibits identical behavior. As established in [11], after discarding conjugate terms, each entry of $\text{FFT}(\mathbf{e}_{\text{real}})$ is asymptotically i.i.d. following distribution W , whose probability density function is given by

$$f_W(z) = \frac{2}{\pi\sigma^2} K_0\left(\frac{2|z|}{\sigma}\right), \quad z \in \mathbb{C}.$$

We only concern ourselves with samples that produce large-magnitude $\mathbf{e}_{\text{real}}$, so we adopt the standard asymptotic approximation $K_0(x) \sim \sqrt{\frac{\pi}{2x}} e^{-x}$ for $x \rightarrow \infty$. Under this approximation, the joint probability density of $\text{FFT}(\mathbf{e}_{\text{real}})$ far from the origin is asymptotically proportional to

$$f_{W'}(z_1, \dots, z_{n/2}) = \left(\prod_i |z_i|^{-1/2} \right) \exp\left(-\sum_i |z_i|\right).$$

The density is dominated by the term $\exp(-\sum_i |z_i|)$, i.e., the ℓ_1 norm of $\text{FFT}(\mathbf{e}_{\text{real}})$. So, our target is to make $\mathbf{e}_{\text{real}}$ distribute within in a ℓ_1 -norm contour, which is an areas defined as $\{\|\mathbf{z}\|_1 \leq T^2 \mid \mathbf{z} \in \mathbb{C}^n\}$, for sufficiently large T . Nevertheless, the support of $f_{W_{n/2}}(\mathbf{z})$ does not exactly coincide with a single ℓ_1 -norm contour; it forms a concave star-shaped region, as illustrated in Figure 4b. We thus apply rejection sampling to eliminate extreme outlier samples concentrated at the thorns of $\text{FFT}(\mathbf{e}_{\text{real}})$, such that nearly all samples retained by the filter lie within some ℓ_1 -norm contour. To be more specific, our procedure is described as follows: for

normally sampled e and r , we introduce a rejection sampling filter that only retains samples satisfying $\|\text{FFT}(e)\|_\infty < T$ and $\|\text{FFT}(r)\|_\infty < T$. Provided T is sufficiently large, almost all filtered samples satisfy $\|\text{FFT}(e \cdot r)\|_1 < T^2$ with high probability.

In addition, for any vector whose ℓ_1 -norm is at most T^2 , its ℓ_2 -norm (length) cannot exceed T^2 , meaning the vector is contained inside the ℓ_2 -ball of radius T^2 . This implies nearly all samples passing the filter yield $\text{FFT}(\mathbf{e}_{\text{real}})$ confined within the radius- T^2 ℓ_2 -ball. We then adopt the i.i.d. complex Gaussian distribution whose samples concentrate around the radius- T^2 hypersphere as the distribution of $\text{FFT}(\mathbf{e}_{\text{vir}})$, which ensures that $\text{FFT}(\mathbf{e}_{\text{vir}})$ is probabilistically larger than $\text{FFT}(\mathbf{e}_{\text{real}})$. Our polynomial ring possesses favorable algebraic properties such that the FFT transformation matrix is a constant multiple of a unitary matrix, which transfers the magnitude bound back to the time domain and guarantees that $\mathbf{e}_{\text{vir}}$ is probabilistically larger than $\mathbf{e}_{\text{real}}$.

We define T to be sufficiently large if the rejection rate induced by the constraint $\|\text{FFT}(e)\|_\infty < T$ is sufficiently small. In our construction of $\mathbf{e}_{\text{vir}}$, we choose T such that the corresponding rejection rate induced by $\|\text{FFT}(e)\|_\infty < T$ is less than λ^{-3} , ensuring a sufficiently conservative choice of $\mathbf{e}_{\text{vir}}$. The recommended $\mathbf{e}_{\text{vir}}$ to construct polar codes is in Table 2,

For practical implementation of the algorithm, we advise against setting the rejection threshold T used in the heuristic for constructing $\mathbf{e}_{\text{vir}}$ as the real rejection threshold T_0 . Instead, we recommend maintaining a practical rejection rate no lower than λ^{-2} . This configuration incurs negligible performance overhead while filtering out far more extreme samples compared to the λ^{-3} rejection rate utilized in our heuristic. In all our subsequent experiments, we consistently adopt a rejection threshold with a rejection rate below 10^{-6} for testing¹. In our final parameter set, we adopt an even more stringent threshold T_0 with a rejection rate fixed at 1% for additional conservatism.

6.3 PolarLAC under Existing DFR Attacks

We evaluate the error-correction capability of PolarLAC via newly designed attacks that maximize correlation. In this section, we demonstrate how to fully exploit the available computational budget to construct strongly correlated noise samples.

Secure DFR model. PolarLAC adopts multi-target protection. So, a safe DFR will satisfy the following inequality:

$$C_{\text{pre}}(e) \cdot C_{\text{pre}}(r) / \text{DFR}^\dagger \leq 2^\lambda, \quad (1)$$

where $\text{DFR}^\dagger$ is the DFR conditional on the corresponding precomputation. To maximize the power of attack, we make full use of all the precomputational budget ² 2^λ for the security level λ . It is observed in all our empirical results that PolarLAC gives sufficiently large DFR margin even in this case.

The best budget allocation strategy. Without loss of generality, we use complex Gaussian $\mathcal{CN}(0, n\sigma^2)$ to denote the distribution of each coordinate of $\text{FFT}(e)$ and $\text{FFT}(r)$. As illustrate in Section 5.4, the correlation will be amplified when e and r has a strong component simultaneously at the same frequency. So, the best attack will maximize $|\text{FFT}_i(e)| \cdot |\text{FFT}_i(r)|$ for some i . Assume a computational budget of $2^{\alpha\lambda}$ and $2^{(1-\alpha)\lambda}$ is allocated to $|\text{FFT}_i(e)|$ and $|\text{FFT}_i(r)|$, respectively, for $\alpha \in (0, 0.5]$. In this setting, since $|\text{FFT}_i(e)|$ and $|\text{FFT}_i(r)|$ follow the same Rayleigh distribution with the inverse of their *erfc* being $\sigma\sqrt{2\alpha\lambda \ln 2}$ and $\sigma\sqrt{2(1-\alpha)\lambda \ln 2}$, the magnitude of $|\text{FFT}_i(e)| \cdot |\text{FFT}_i(r)|$ scales linearly with $(\alpha(1-\alpha))^{1/2}\lambda$. It reaches its maximum when $\alpha = 0.5$. So, allocating the computation cost equally to search for bad e and r will worsen the channel to the utmost.

The decryption error includes 4 polynomial products i.e., $h' = e \cdot r + e' \cdot r' - s \cdot e_1 - s' \cdot e'_1$. The best strategy to allocate precomputational budget over the 4 summands is to concentrate all

¹ This value is smaller than λ^{-2} for all practical choices of security levels.

² We define the *precomputation* as searching for both e and r .

computational resources on a single term. The reasoning is provided below. The i -th frequency bin of $\text{FFT}(h')$ is upper bounded by

$$\begin{aligned} & |\text{FFT}_i(e \cdot r) + \text{FFT}_i(e' \cdot r') - \text{FFT}_i(e_1 \cdot s) - \text{FFT}_i(e'_1 \cdot s')| \\ & \leq \sum |\text{FFT}_i(x \cdot y)| \text{ for } (x, y) = \{(e, r), (e', r'), (e_1, s), (e'_1, s')\} \end{aligned}$$

If the budget allocated to the j -th summand is 2^{λ_j} , then the inverse of erfc of RHS under the corresponding budget is proportional to $\sum_{j=1}^4 \lambda_j = \lambda$. However, the phase of $\text{FFT}_i(x \cdot y)$ is uniform. If we distribute the budget over the four summands, the inverse of erfc of the LHS will no longer match that of the RHS, unless additional budget is consumed to align their phases. So, the most efficient way is to dedicate to a single summand.

DFR under an Improved Attack. In LAC, there are cases where only a fraction of the ciphertext c_2 conveys the plaintext. So, the adversary may concentrate its budget on the informative segment of the ciphertext. In PolarLAC, every coefficient of ciphertext c_2 is informative. So, we take the l_2 -norm $\|h\|$ as a indicator of channel quality.

The Type-2 pattern in [22] is defined by an interval of length $L_0 + L_1$ with at least L_1 either all positive or all negative entries and the remaining entries all-zero. We found that Type-2 gives stronger zero-frequency component than Type-1, see Fig. 6. Nonetheless, we need to search for the most powerful Type-2 over all possible L_0 and L_1 . We give an example of how the precomputational budget varies with L_0 and L_1 in Fig. 7. Moreover, for the same C_{pre} , we depict how the infinity norm of the magnitude of $\text{FFT}(e_1)$ varies with L_1 in Fig. 8, as well. It turns out that for the same C_{pre} , a bigger L_1 gives a stronger zero-frequency component. So, from the adversary's perspective, Type-2 with bigger L_1 is a better choice. However, for multiple choices (L_0, L_1) achieving the same C_{pre} , we opt to select a smaller L_0 because more zeros will suppress the existence of other strong frequency component.

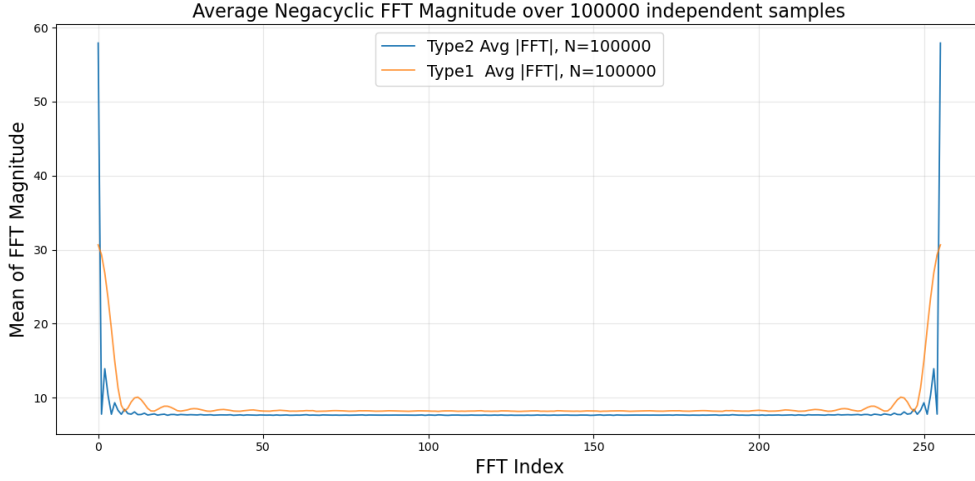


Fig. 6: Power spectrum of Type-1 and Type-2 with $C_{pre} = 2^{64}$ for PolarLAC-128

In [22], an adversarial s satisfies the property that

$$\left| \sum_{i=0}^{n-1} s_i \right| \geq \delta_0,$$

for a positive integer δ_0 . We in this document refer it as Type-S. It is reasonable to approximate $\sum_{i=0}^{n-1} s_i$ by a Gaussian distribution by central limit theorem. The budget to encounter such s is

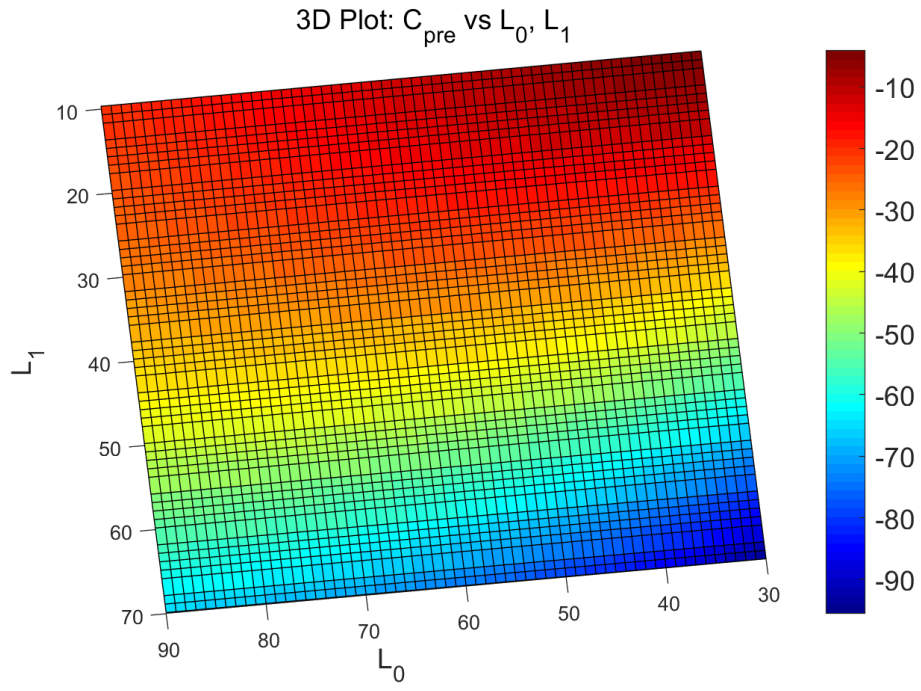


Fig. 7: C_{pre} v.s. (L_0, L_1) for PolarLAC-128

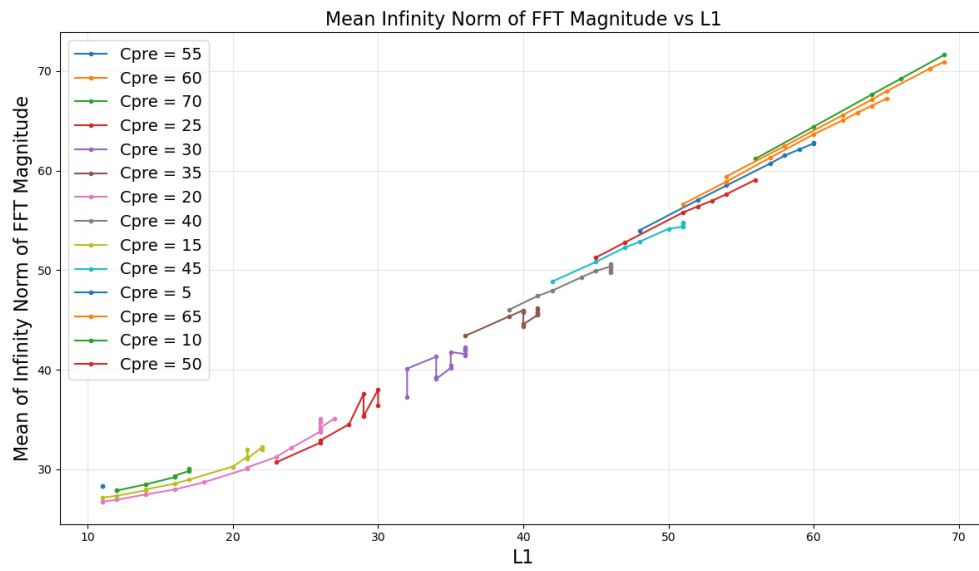


Fig. 8: The magnitude of $\text{FFT}_1(e)$ v.s. L_1 for PolarLAC-128

the erfc when it takes values beyond the interval of $(-\delta_0, +\delta_0)$. Based on the empirical results, Type-S is moderately stronger in terms of spectral norm and the ℓ_2 -norm than Type-2 at the same $C_{pre} = 2^{\lambda/2}$, see Fig. 9.

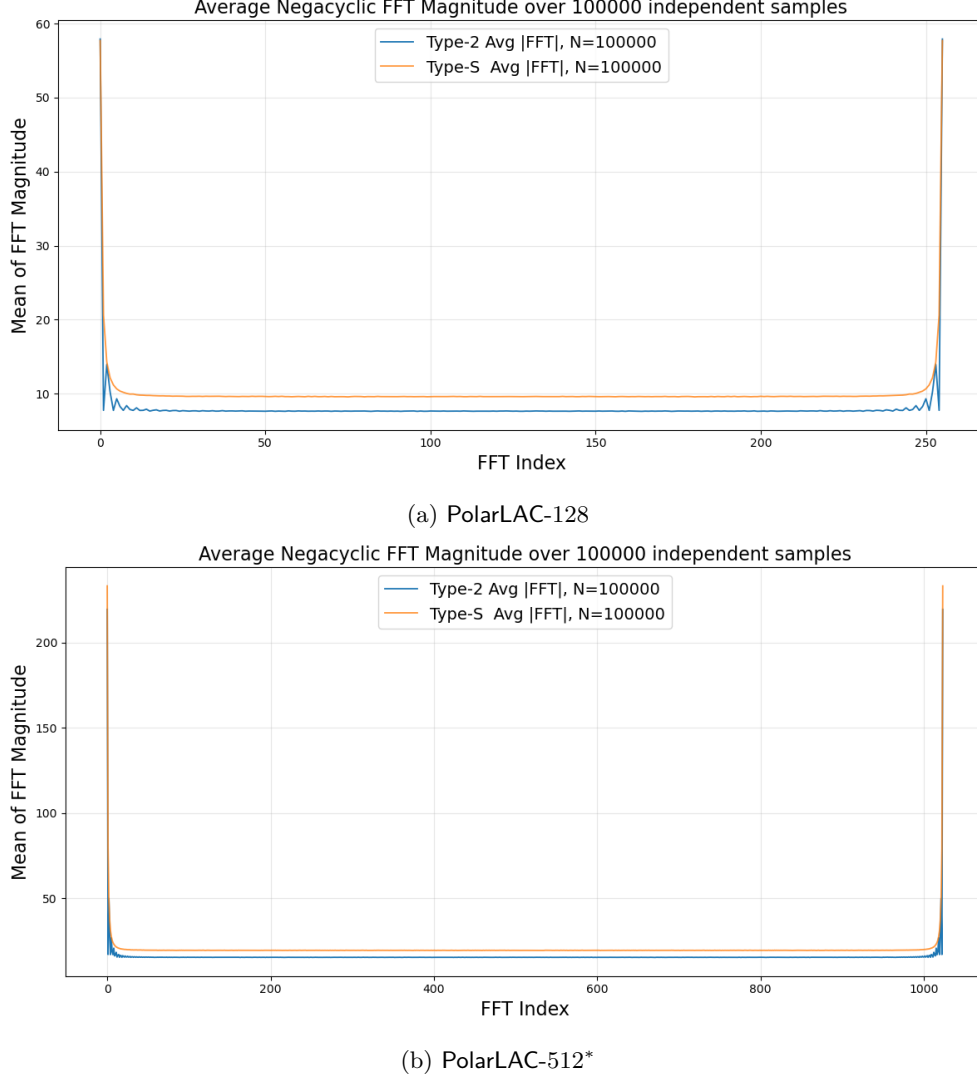


Fig. 9: Type-2 vs Type-S

Shall the adversary devise an attack using the existing patterns, the correlation could be exploited to the utmost if (a) precomputational budget is concentrated on a single summand of $h' = e \cdot r + e' \cdot r' - s \cdot e_1 - s' \cdot e'_1$, e.g. $e \cdot r$ (b) both e and r are Type-S with equal precomputational budget of $2^{\lambda/2}$.

For safety conservatism, we explore the DFR margin under a resource exhaustion attack. We simulated the h' with the 1st summand to be the product of two Type-S with identical $C_{pre} = 2^{\lambda/2}$, and the rest summands are sampled via the standard routine.

PolarLAC demonstrates the error-correction performance listed in Table 4. To further probe its performance limits, we consider a resource exhaustion attack scenario where the adversary could arbitrarily apply precomputation to every secret information e, r, e', r' , and so on. First of all, we found that the adversarial samples are all rejected by the spectral rejection sampling using the predefined bigger threshold T in Table 2. Secondly, we carried out additional experiments

showing that PolarLAC-Light, -128, -256 and -512* are capable to correct the errors even if we test it with the rejected samples. For PolarLAC-512, its DFR of 2^{-323} is not aligned with its security category. This accounts for the occurrences of decryption failures under a resource exhaustion attack.

category	budget	# rejections / 10^5 samples	# dec. failures / 10^5 samples	$\mathbb{E}(\# \text{ bit errors})$ / sample)	$\max(\# \text{ bit errors})$ / sample)
PolarLAC-Light	2^{128}	10^5	0	0.0	0
PolarLAC-128	2^{128}	10^5	0	0.04	3
PolarLAC-256	2^{256}	10^5	0	0.70	12
PolarLAC-512	2^{512}	10^5	1252	17.76	84
PolarLAC-512*	2^{512}	10^5	0	0.02	3

budget : $C_{pre}(e) \cdot C_{pre}(r)$
 #dec. failures : number of decryption failures of PolarLAC **using the rejected** h
 #bit errors : bit errors **without** polar codes **using the rejected** h'
 $\mathbb{E}, \max$: statistics of # bit errors per **rejected** sample of h'

Table 4: Capability of error correction of PolarLAC under a **resource exhaustion attack** of Type-S (10^5 iterations).

6.4 PolarLAC under a Novel DFR Attack

To maximize decryption failure in absence of rejection sampling. Recall the conclusions from [11]: asymptotically, the non-conjugate components of $\text{FFT}(e \cdot r)$ are i.i.d., and the correlation between entries of $e \cdot r$ is tightly coupled with its ℓ_2 -norm. Specifically, stronger correlation among sample entries corresponds to a larger ℓ_2 -norm, and conversely, samples with large magnitudes exhibit extremely strong correlation across their coordinates with high probability.

In short, any adversary aiming either to exploit the incompletely characterized correlation within $e \cdot r$, or to generate large-magnitude $e \cdot r$ noise that maximizes decoding failures of the scheme, will optimally perform precomputation to maximize the magnitude of $e \cdot r$ which is proportional to that of $\text{FFT}(e \cdot r)$. In this section, we significantly strengthen the adversarial power model: we directly evaluate the error-correction performance of the polar code adopted in PolarLAC against noise induced by low-probability extreme samples of $\text{FFT}(e \cdot r)$. Under this adversarial model, the adversary can convert its full computational budget 2^λ into a guarantee that PolarLAC always encounters extreme noise events with probability $2^{-\lambda}$. This stands in contrast to the realistic setting, where the adversary must repeatedly query new users to target secret information e and perform massive precomputation to acquire extreme r , before it can produce maximally extreme $\text{FFT}(e \cdot r)$ realizations.

Since the entries of the non-conjugate components of $\text{FFT}(e \cdot r)$ are i.i.d., and extreme samples concentrate near the complex coordinate axes as demonstrated in [11], the optimal adversarial strategy is to maximize the magnitude of a single entry within $\text{FFT}(e \cdot r)$ while keeping the remaining entries at their typical average values. Concretely, we seek the minimal threshold ℓ satisfying

$$\frac{n}{2} \cdot \Pr[|\text{FFT}_1(e \cdot r)| \geq \ell] \leq 2^{-\lambda}.$$

The prefactor $n/2$ accounts for the adversary's freedom to select any one of the $n/2$ frequency bins; a single sufficiently large entry suffices, and the target does not have to be the first frequency component. As derived in [11], the magnitude $|\text{FFT}_1(e \cdot r)|$ follows the marginal probability density function

$$f_{|W|}(r) = \frac{4r}{\sigma^4} K_0\left(\frac{2r}{\sigma^2}\right), \quad r \geq 0,$$

where σ denotes the standard deviation of $\text{FFT}_1(e)$.

Our procedure for generating extreme samples proceeds as follows: we fix $|\text{FFT}_1(e \cdot r)| = \ell$, sample a uniform phase to construct the complex value $\text{FFT}_1(e \cdot r)$, and draw all other frequency entries and auxiliary terms according to their typical average distributions.

To maximize decryption failure in presence of rejection sampling. The above described strategy will fail when rejection sampling filter is employed. But we highlight that this frequency-domain sampling paradigm exhibits strong extensibility and can be naturally adapted to construct practical attacks that account for the rejection sampling filters applied to e and r . Recall our rejection threshold constraint $\|\text{FFT}(e)\|_\infty < T$, which imposes a natural upper bound T^2 on the magnitude of any single component $|\text{FFT}_1(e \cdot r)|$. Since all entries of $\text{FFT}(e \cdot r)$ share an identical marginal distribution and are asymptotically independent, we may allocate a separate computational budget 2^{λ_i} to a certain frequency component $\text{FFT}_i(e \cdot r)$. This allows us to design the optimal precomputation strategy under the rejection threshold constraint, maximizing the probability of triggering a decoding error.

Our frequency-domain sampling method is specifically designed for this threshold-constrained scenario. When generating samples, we grant the adversary every computational precision advantage possible. Meanwhile, we adopt the looser threshold T with a rejection rate below 10^{-6} —rather than the stricter practical threshold T_0 with a 1% rejection rate. This configuration serves two purposes: first, it validates the soundness of our heuristic for selecting the virtual stronger noise; second, it ensures that real-world operating conditions for the adversary are strictly worse than our experimental setup, yielding an additional conservative safety margin.

The optimized frequency-domain sampling proceeds as follows. Let T denote the rejection threshold for $|\text{FFT}_i(e)|$ and $|\text{FFT}_i(r)|$. Define W_T as the random variable corresponding to $|\text{FFT}_i(e \cdot r)|$ generated solely from samples passing the threshold filter. We assume an adversary who makes full use of all the precomputational budget to explore the DFR margin of PolarLAC. A naive strategy of expending the entire budget to push a single $|\text{FFT}_i(e \cdot r)|$ close to the hard upper bound T^2 wastes substantial computational power. Redistributing the wasted budget across additional frequency bins yields a noise vector with a larger overall magnitude. We therefore model the adversary’s strategy as uniform budget allocation across k distinct frequency bins to produce extreme samples. For each candidate k , we estimate the total magnitude $\|\text{FFT}(e \cdot r)\|$ and solve for the optimal value k_{best} that maximizes this norm within the computation budget 2^λ .

We observed that for any $k < k_{\text{best}}$, fully allocating the available computational resources forces the k targeted bins to saturate near the bound T^2 . This confirms that the optimal sampling strategy under rejection filtering is to amplify k_{best} frequency bins simultaneously. To adopt an even more conservative evaluation criterion and guarantee our generated samples are more extreme than any uneven budget allocation across k_{best} bins, we construct $\text{FFT}(e \cdot r)$ by directly assigning a magnitude of T^2 with uniform random phase to its k_{best} entries, while sampling all remaining entries and components according to their typical average distributions. Finally, we treat the resulting vector as additive noise, pair it with randomly selected messages, and feed the combined signals into our error-correcting code for experimental testing.

The procedure for computing k_{best} is elaborated as follows. For any integer k , we evenly distribute the total computational budget across k arbitrary entries of $\text{FFT}(e \cdot r)$. Analogous to our prior derivation, we solve for the corresponding threshold ℓ_k satisfying

$$\Pr [|\text{FFT}_1(e \cdot r)| \geq \ell_k] \leq \left(\binom{n/2}{k}^{-1} \cdot 2^{-\lambda} \right)^{1/k}.$$

The binomial coefficient $\binom{n/2}{k}$ appears here to account for the adversary’s freedom to choose any k frequency positions. We then fix the magnitude of the k selected entries of $\text{FFT}(e \cdot r)$ to ℓ_k with uniformly random phases, sample all other entries randomly, and compute the resulting ℓ_2 norm of the extreme sample. By iterating over $k = 1, 2, \dots$, we identify k_{best} , the value that maximizes $\|\text{FFT}(e \cdot r)\|$. Calculating k_{best} fundamentally relies on the cumulative distribution

function (CDF) of W_T , which is given as

$$\Pr[W_T \leq r] = c \cdot \left[1 - \exp\left(-\alpha \cdot \frac{r^2}{T^2}\right) \right] + c^2 \cdot \left[\exp\left(-\alpha \cdot \frac{r^2}{T^2}\right) - \exp(-\alpha \cdot T^2) \right] - 2c^2 \alpha \exp(-2\alpha \cdot r) \cdot I(r),$$

where

$$I(r) = \int_{r/T}^T x \cdot \exp[-\alpha \cdot (x - r/x)^2] dx,$$

and $c = \frac{1}{1 - \exp(-T^2/n\sigma^2)}$, $\alpha = 1/n\sigma^2$, and σ is the standard deviation per coefficient of e, r .

In Table 5, we demonstrate the correctness of PolarLAC-128, -256, and -512 with the extremely bad samples generated by our frequency-domain sampling. We only observed decryption failures for PolarLAC-512. **However, in the experiment setting, if the adversary does find h' having k_{best} frequency components with their magnitude close to T^2 , it would takes a budget of at least 2^{572} . In fact, if the budget of 2^{512} is distributed evenly to the k_{best} many frequency components, no failures are observed in our experiment.**

category	budget	k_{best}	# dec. failures / 10^5 samples	$\mathbb{E}(\# \text{ bit errors}$ / sample)	$\max(\# \text{ bit errors}$ / sample)
PolarLAC-128	2^{128}	3	0	0.19	5
PolarLAC-256	2^{256}	4	0	0.83	12
PolarLAC-512	2^{512}	10	4	0.6	22
PolarLAC-512*	2^{512}	6	0	0.04	5

budget : $C_{pre}(e \cdot r)$
 k_{best} : h satisfies k_{best} frequency components s.t. $|\text{FFT}_i(e \cdot r)| = T^2$
 #dec. failures : number of decryption failures of PolarLAC
 #bit errors : bit errors **without** polar codes
 $\mathbb{E}, \max$: statistics of # bit errors per sample of h'

Table 5: Capability of error correction of PolarLAC under a **resource exhaustion** attack from frequency domain (10^5 iterations)

6.5 Alternative Rejection Sampling via Autocorrelation Polynomial

Our error-correction framework is built upon a rejection threshold T such that the rejection probability conditioned on $|\text{FFT}_i(e)| < T$ is below 10^{-6} . This condition seemingly requires us to compute the FFT of e before carrying out threshold checks. To generalize our construction to extensive software/hardware architectures, we also propose a procedure which realizes $|\text{FFT}_i(e)| < T$ without really implementing the FFT operations. This section presents a time-domain rejection scheme that satisfies the constraint using only time-domain arithmetic.

Let $e \in R_q$ denote an n -dimensional polynomial, and let $\bar{e}$ stand for its conjugate polynomial. Two equivalent methods exist to compute $\bar{e}$. The first still relies on FFT transformation: $\bar{e} = \text{FFT}^{-1}(\overline{\text{FFT}(e)})$. The second approach requires no FFT at all: $\bar{e} = e(x^{-1})$. Explicitly, treat e as a polynomial, substitute the indeterminate x with x^{-1} , then reduce the resulting expression modulo the defining polynomial $x^n + 1$ to retrieve the coefficient representation of $\bar{e}$. We filter e by retaining only samples satisfying $\|e \cdot \bar{e}\|_1 \leq T^2$, i.e., we constrain the ℓ_1 -norm of the coefficients of the product polynomial $e \cdot \bar{e}$. Any e passing this filter is guaranteed to satisfy $|\text{FFT}_i(e)| < T$ for all frequency indices i , and the polynomial multiplication can be accelerated via standard transforms such as NTT.

We now prove that any polynomial e obeying $\|e \cdot \bar{e}\|_1 = \sum_j |(e \cdot \bar{e})_j| \leq T^2$ must satisfy $|\text{FFT}_i(e)| < T$ for every i . The derivation proceeds as follows:

$$|\text{FFT}_i(e)|^2 = |\text{FFT}_i(e) \cdot \overline{\text{FFT}_i(e)}| = |\text{FFT}_i(e \cdot \bar{e})| = \left| \sum_{j=0}^{n-1} (e \cdot \bar{e})_j \cdot \zeta_i^j \right| \leq \sum_j |(e \cdot \bar{e})_j| \leq T^2.$$

The first inequality follows from the triangle inequality, then immediately yields $|\text{FFT}_i(e)| < T$.

Naturally, the acceptance condition $\|e \cdot \bar{e}\|_1 \leq T^2$ constitutes a stricter constraint in essence, which inevitably raises the overall rejection rate. According to our empirical tests, when the rejection rate induced by $|\text{FFT}_i(e)| < T$ is below 10^{-6} , the rejection rate corresponding to the condition $\|e \cdot \bar{e}\|_1 \leq T^2$ falls roughly at the order of 1%.

7 Security Analysis

7.1 Theoretical Security

The requirements for IND-CPA and IND-CCA security are standard. In the IND-CPA security game, any efficient adversary that can obtain encryptions of messages of its choice, could not distinguish the ciphertext of one message from the other. In the IND-CCA security game, the adversary can additionally query a decryption oracle on ciphertexts of its choice, except for the exact challenge ciphertext. We model the employed hash functions as random oracles, to which the adversary may have quantum access.

Reduction for the PolarLAC.PKE.CPA Scheme. IND-CPA security can be obtained straightforwardly as that of [27].

Theorem 2. *If the MLWE assumption with $\chi = \Psi_\sigma$ holds, our PolarLAC.PKE.CPA constructed in 4.1 is IND-CPA secure.*

Proof. Since the MLWE_{Rej} assumption can be reduced to the standard MLWE assumption, it suffices to reduce the IND-CPA security of PolarLAC.PKE.CPA to MLWE_{Rej} .

We first replace the public-key component $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$ with a uniformly random ring vector that passes the rejection test. This modification is indistinguishable under the MLWE_{Rej} assumption. Note that the reduction is given $\mathbf{A}$, and hence can generate an $\mathbf{A}$ distributed according to RejSampU from an initially uniform matrix. Similarly, the reduction can test whether $\mathbf{b}$ lies in the required range and output only those instances that pass the rejection test.

Next, we replace $(\mathbf{c}_1, \mathbf{c}_2)$ in the challenge ciphertext with uniformly random values. The indistinguishability of this modification is again guaranteed by the MLWE_{Rej} assumption. After this step, $\mathbf{c}_2$ is uniformly distributed over $\mathbb{Z}_{2^d}^n$, and therefore information-theoretically hides the encapsulated message. $\square$

Remark. The MLWE_{Rej} problem filters samples from MLWE instances without any knowledge of the secret. Note that hardness proofs for standard MLWE hold for any polynomial number of available samples. Hence, as long as the acceptance rate remains bounded, the hardness of MLWE_{Rej} is equivalent to that of vanilla MLWE. The full formal definition of MLWE_{Rej} is provided in Section 2.3.

For $q = 257$, we retain only MLWE samples $(\mathbf{a}, b)$ such that no coefficient of any polynomial entry in $\mathbf{a}$ equals 256, and the total count of coefficients equal to 0 or 256 within polynomial b does not exceed γ . A detailed analysis of the corresponding rejection rate is given in Section 5. Under our parameter setup, a trivial reduction from standard MLWE to MLWE_{Rej} holds unconditionally.

By contrast, no sample filtering is required for MLWE instances when $q = 769$, meaning our scheme directly relies on the standard MLWE problem in this case.

Reduction for the PolarLAC.KEM.CCA Scheme. Since PolarLAC.KEM.CCA is obtained by a standard application of the (salted) FO transformation with implicit rejection to PolarLAC.PKE.CPA, the decryption failure probability is negligible according to 3. IND-CCA security can be obtained according to the result of [25,18].

Theorem 3. *If the MLWE assumption with $\chi = \Psi_\sigma$ holds, the size of the message space $\mathcal{M}$ of PolarLAC.PKE.CPA is super-polynomial, the decryption failure probability of PolarLAC.PKE.CPA is negligible, then our PolarLAC.KEM.CCA constructed in 4.2 is IND-CCA secure in the quantum random oracle model.*

7.2 Practical Security

Following the methodology used in the NIST PQC standardization documents for lattice-based schemes, our concrete security estimation treats the underlying Module-LWE instances as coefficient - level LWE instances and estimates the cost of the best known attacks on these induced instances. This gives a common and conservative basis for comparison with other lattice-based submissions: the estimation does not rely on unproved additional hardness coming from the module structure, while known algebraic attacks are considered separately when they apply. The resulting bit-security values for each PolarLAC parameter set are reported in Table 3; each entry gives the classical cost and the corresponding quantum cost, denoted by “Q”.

In our concrete security estimation, we instantiate the induced LWE instances in the Lattice-Estimator with the parameters listed in Table 3. In particular, the module rank k , the ring dimension n , the modulus q , and the secret/error distribution are taken directly from the corresponding columns of the table. The resulting coefficient-level LWE dimension is $N = kn$. It remains to specify the number of available samples m , which is determined by the ciphertext structure as follows.

We set m according to the number of coefficient-level LWE equations available from the ciphertext. As specified in Section 4.1, the first ciphertext component $\mathbf{c}_1$ lies in $R_q^{k \times 1}$, and hence consists of k ring elements. It therefore contributes kn coefficient-level LWE samples. The second ciphertext component is stored as $\mathbf{c}_2 \in \mathbb{Z}_{2^d}^n$, and corresponds to one underlying ring element before compression. Hence it contributes n coefficient-level equations. The compression of this component affects the rounding error and the ciphertext byte length, but it does not change the number of coefficient positions available from this ring element. Since all recommended parameter sets in Table 3 take $k = 2$, the contribution from $\mathbf{c}_2$ is $n = kn/2$. Consequently, the total number of samples used in the concrete security estimation is

$$m = kn + n = (k + 1)n = \frac{3}{2}kn,$$

rather than $2kn$. Here m denotes the number of coefficient-level LWE samples supplied to the estimator, and should not be confused with the byte length of the ciphertext, which is additionally affected by the compression parameters d_1 and d_2 .

We used the Lattice-Estimator [1,2] (commit `3e48ef4`) to estimate all proposed parameter sets. The estimator implements the standard attack families currently used for LWE-type schemes, including primal lattice attacks, dual lattice attacks, hybrid variants combining lattice reduction with guessing, and the non-lattice attacks enabled in the estimator, such as BKW-type attacks and Arora-Ge algebraic attacks when their preconditions are relevant. In our estimation, all of these attacks were enabled and compared under the same parameter description of the PolarLAC instances.

The BKZ cost estimates were computed under two cost models. The first is the core-SVP model, implemented in the estimator as the ADPS16 cost model, which measures the cost of BKZ reduction by the cost of solving the exact SVP subproblems in the BKZ blocks [4,8]. Concretely, once the estimator determines the BKZ block size β needed for a successful attack, the classical and quantum costs are derived from the best known asymptotic costs of lattice sieving for dimension β , as in the usual NIST-style estimates for LWE and module-/ring-LWE schemes. This model is intentionally simple and allows direct comparison with earlier estimates based on BKZ plus classical or quantum sieving.

The second model is the refined BKZ model, implemented in the estimator as the **MATZOV** cost model. In this model, the cost of the full BKZ reduction is estimated more explicitly, including the effect of tours, sieving calls, success probability, and the dimension of the concrete embedding or dual lattice considered by the attack [3,7,12,31]. In particular, the refined BKZ model incorporates the refined lattice-reduction cost estimates used in the Kyber documentation and the quantum/classical sieving-cost analysis of Albrecht et al., with the list-decoding improvement reported in the MATZOV security report. It is therefore less coarse than the core-SVP approximation. We report both models because the concrete cost of very large-block BKZ is not a settled engineering fact; using both gives a clearer view of the margin of the parameters.

After applying these two cost models to all attacks implemented in the estimator, the best estimates for **PolarLAC** are obtained from lattice-reduction based attacks. More precisely, under the core-SVP model the lowest costs are given by the **dual_hybrid** attack (**matzov**) together with the **usvp** attack, while under the refined BKZ model the lowest costs are given by the **dual_hybrid** attack (**matzov**) together with the **bdd** attack [31]. The remaining attack families do not give a lower security estimate for our parameters. The difference between the winning attacks in the two models comes from the optimization of the **bdd** parameters: the best balancing point for **bdd** changes when the BKZ cost model is changed, making **bdd** slightly worse than **usvp** in the core-SVP accounting but slightly better in the refined BKZ accounting. The numerical gap between these two attacks is small, so this should be viewed as a model-dependent tie rather than as evidence of a qualitatively different attack surface.

For the 512-bit security level, i.e., the last two rows in Table 3, the required BKZ block sizes exceed the default search range of the estimator. The default value of **max_beta** in **conf.py** is 1754, which prevents the program from returning estimates whose optimal block size is above this bound. In our estimation of the 512-bit parameter sets, we changed this value from 1754 to 3000. This modification does not change the attack formulas or cost models; it only enlarges the block-size search interval so that the estimator can return security values above 512 bits.

The same estimation gives margins for the lower categories as well. These estimates include the best attack selected by the estimator among its implemented candidates, rather than only a single hand-picked attack. In particular, the sparse ternary distributions of secret and error are also accounted for through the estimator’s distribution and hybrid-attack machinery.

We stress that these numbers are practical concrete security estimates, not formal reductions. They rely on the current state of knowledge about BKZ, sieving, and LWE cryptanalysis, and should be interpreted in the same way as the estimates used in NIST PQC lattice submissions such as Kyber [7]. Within this framework, the selected **PolarLAC** parameters meet their intended classical security targets and keep a comfortable margin against the best known quantum attacks implemented by the estimator.

8 Implementation and Performance

We provide both a reference implementation and an optimized implementation on the x86 platform. The reference implementation is written in portable C and does not rely on platform-specific instruction sets or compiler-assisted vectorization. It is intended to provide a clear and reproducible baseline for verifying the correctness of the proposed KEM.

The optimized x86 implementation is developed using the AVX2 instruction set. In this version, the main arithmetic components, including polynomial multiplication, sampling-related operations, and compression routines, are optimized by exploiting SIMD parallelism. This implementation is designed to evaluate the practical performance of the scheme on modern x86 processors.

8.1 Polynomial Multiplication

In the MLWE implementation, the dominant algebraic operations are defined over the module structure induced by the polynomial quotient ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. The public matrix, secret vector, and error vector are composed of polynomials in R_q , and the module rank is denoted by

k . Therefore, the core computations in key generation, encryption, and decryption are matrix-vector multiplications and vector inner products over R_q . To accelerate these operations, each component-wise polynomial multiplication is implemented using an NTT-based method.

NTT Structure. The root of unity structure provided by the selected modulus is limited, so a fully expanded negacyclic NTT cannot be directly applied for all parameter sets. For example, the modulo 257 and 769 can natively support only a full radix-2 NTT of length at most 128, a fully expanded single NTT cannot be applied directly to the entire polynomial when the polynomial dimension is $n = 256, 512, 1024$. To address this issue, the implementation adopts a partially expanded NTT strategy, where the unsupported part is retained as small local convolutions of length B . We denote such a local base multiplication by $P_n(B)$, which means that each frequency component in the NTT domain is associated with a local polynomial multiplication of length B . To improve the efficiency of base multiplication by $P_n(B)$, we use the Karatsuba method to reduce the complexity of the base multiplications. The moduli, transform layers, and local convolution sizes for different parameter sets are summarized in Table 6.

Parameter set	n	q	Unexpanded layers	(I)NTT layers	Local convolution size
PolarLAC-Light	256	257	1	7	$P_{256}(2)$
PolarLAC-128	256	257	1	7	$P_{256}(2)$
PolarLAC-256	512	257	2	7	$P_{512}(4)$
PolarLAC-512	1024	257	3	7	$P_{1024}(8)$
PolarLAC-512*	1024	769	3	7	$P_{1024}(8)$

Table 6: Partially expanded NTT structure for the MLWE parameter sets

At the module level, the main operation in key generation is the multiplication between the public matrix and the secret vector. The public matrix contains $k \times k$ polynomial entries, and the secret vector contains k polynomial entries. Therefore, the matrix-vector multiplication consists of several component-wise polynomial multiplications together with the corresponding polynomial additions. During encryption, the ciphertext vector is generated by multiplying the transpose of the public matrix with the random vector, while the scalar ciphertext component is obtained from the inner product between the public-key vector and the random vector. During decryption, the inner product between the ciphertext vector and the secret vector is computed to recover the polynomial used for message decoding. These matrix-vector multiplications and vector inner products are organized around the NTT-domain representation to avoid high-complexity coefficient-domain convolutions.

To reduce the number of transforms of NTT and INTT in the whole procedure, we store the public key and secret key in the NTT domain. Specifically, in PolarLAC-512* encryption, to avoid the effect of the Montgomery factor, the ciphertext vector $\mathbf{u} = A^T \mathbf{r} + \mathbf{e}_1$ is generated directly in the NTT domain and kept in the NTT-domain representation. To match this representation, the error polynomials in $\mathbf{e}_1$ are first transformed into the NTT domain and then added to the result of $A^T \mathbf{r}$. The vector $\mathbf{u}$ is then compressed and stored using a CRT-based representation. In contrast, the scalar ciphertext component $v = \mathbf{b}^T \mathbf{r} + e_2 + \mu$ is still transformed back to the coefficient domain through an inverse NTT, because message embedding, rounding, and subsequent decoding are performed in the coefficient domain.

Reduction Strategy. Moreover, the modulus 257 and 769 are both not larger than 10 bits, and the associated NTT involves only 7 layers, which makes the range of intermediate values in the butterfly operations easy to control. In the concrete implementation, the addition and subtraction results in each butterfly are handled together with the subsequent multiplication stage via lazy reduction. Concretely, intermediate values of the form $a_i \pm \text{MontgomeryReduce}(a_j \cdot \phi)$ are kept within the range of `int16_t`, where the symbol ϕ denotes a unity root. And their modular reduction is completed later through Montgomery reduction during the vectorized pointwise

multiplication phase of the NTT. In this way, explicit modular reduction after every butterfly addition and subtraction can be avoided, so that most modular reduction operations in the transform are concentrated in the multiplication stage. This implementation strategy effectively reduces the overhead in butterfly operations and further improves the overall efficiency of polynomial multiplication.

8.2 Integer FFT for Spectral Rejection

The sampling procedure of secret and error vectors requires a spectral norm check for candidate small polynomials. To improve portability across different platforms, we implement this check using an integer FFT rather than relying on floating-point arithmetic. This design is particularly suitable for resource-constrained chips and embedded platforms, where efficient hardware floating-point units may be unavailable or where floating-point operations may introduce additional area, power, and timing overhead. As we perform FFT on polynomials sampled from small and sparse ternary distributions, we can let the FFT take `int16_t` polynomial coefficients as input, and represents frequency-domain values as 16-bit complex integers. The twiddle factors are precomputed as an integer table and scaled by 2^8 . During twiddle multiplication, intermediate products are accumulated in 32-bit integers and scaled back by rounded right shifts. Since the input polynomial is real-valued, the implementation computes only the unique half-spectrum and recovers the corresponding spectral information using conjugate symmetry.

x86 reference implementation. The x86 reference implementation follows a portable C design and does not rely on platform-specific vector instructions. It uses the integer fixed-point representation for inputs, twiddle factors, and intermediate complex values. The FFT is implemented as a fixed-size half-spectrum transform. Instead of using a generic recursive FFT with runtime length checks at every recursion level, the transform length is specialized at compile time for the supported parameter set. This preserves the standard radix-2 decomposition while removing unnecessary branch overhead from the recursive path. The complex multiplication by twiddle factors is implemented with scalar 32-bit arithmetic, followed by rounded shifting to control the output range. The spectral bound check is also performed in scalar C by accumulating the rejection condition over all unique frequency components.

x86 AVX2 optimized implementation. The optimized x86 implementation further restructures the integer FFT to better match AVX2 SIMD operations. First, the two halves of the real input polynomial are folded into complex 16-bit integer values and placed in bit-reversed order. Then, the transform is evaluated as $\log_2 \frac{n}{2}$ iterative out-of-place radix-2 stages over aligned buffers. This regular iterative schedule is more suitable for aligned AVX2 loads and stores than the recursive layout.

The main AVX2 optimization is the complex twiddle multiplication. Instead of expanding the complex product into several scalar-style 32-bit multiplications, the optimized implementation uses `vpmaddwd`-style 16-bit multiply-add operations. For each vector of complex values, the real and imaginary parts are computed as

$$\text{Re}(t) = x_{\text{re}}\omega_{\text{re}} - x_{\text{im}}\omega_{\text{im}}, \quad \text{Im}(t) = x_{\text{re}}\omega_{\text{im}} + x_{\text{im}}\omega_{\text{re}}.$$

This matches the pairwise 16-bit dot-product structure supported by AVX2 and reduces the number of vector instructions in the FFT combine step. The implementation processes multiple complex values in parallel, applies rounded shifts for fixed-point scaling, and packs the results back into 16-bit complex form.

In the rejection-only path, the optimized implementation also fuses the final FFT stage with the spectral bound check. Instead of materializing the complete half-spectrum and then scanning it, the last-stage outputs are immediately squared and compared with the bound using AVX2 integer operations. The comparison results are accumulated into a vector rejection mask, and the polynomial is accepted only when no component exceeds the bound. This fused design reduces memory traffic and removes an extra pass over the spectrum.

8.3 Efficient Sampling

The sampling procedure in this implementation consists of two main parts: public matrix sampling and small secret/error vector sampling. All samples are deterministically expanded from seeds through a unified XOF interface. Matrix indices or nonces are used for domain separation, so that the pseudorandom streams for different polynomial components remain independent.

Public matrix sampling. The public matrix A is expanded from the public seed. For the (i, j) -th polynomial entry of the matrix, two index bytes are appended to the input seed as domain-separation information. When the transposed matrix A^T is required, the two indices are swapped. For the parameter sets with modulus $q = 257$, each matrix coefficient is generated directly from one XOF output byte, producing coefficients in the range $[0, 255]$. This byte-oriented sampling method avoids coefficient-wise rejection sampling and additional 9-bit unpacking, and therefore has low implementation overhead. For PolarLAC-512 with modulus $q = 769$, the public matrix is sampled using a base- q grouped rejection method. Each group of five coefficients is expanded from a 48-bit random integer smaller than 769^5 , while the remaining group of four coefficients is expanded from a 39-bit random integer smaller than 769^4 . Random integers exceeding the corresponding bounds are rejected, which ensures that the generated coefficients are unbiased and uniformly distributed over $\mathbb{Z}_{769}$.

Small secret and error vector sampling. The small polynomials in the secret vector, error vector, and encryption randomness are sampled from ternary distributions. For PolarLAC-Light and PolarLAC-256, each coefficient is generated from three random bits b_0, b_1, b_2 as $a_i = (b_0 - b_1)b_2$, which gives

$$\Pr[a_i = -1] = \frac{1}{8}, \quad \Pr[a_i = 0] = \frac{3}{4}, \quad \Pr[a_i = 1] = \frac{1}{8}.$$

For PolarLAC-128 and PolarLAC-512, each coefficient is generated from four random bits as $a_i = (b_0 \wedge b_1) - (b_2 \wedge b_3)$, which gives

$$\Pr[a_i = -1] = \frac{3}{16}, \quad \Pr[a_i = 0] = \frac{5}{8}, \quad \Pr[a_i = 1] = \frac{3}{16}.$$

This bit-oriented ternary sampler only requires shifts, bitwise operations, and additions/subtractions, avoiding table lookups or costly discrete-distribution sampling. To control the decryption-failure probability, some small polynomials are additionally filtered after generation. The implementation first samples a candidate polynomial from the seed and the current nonce, and then checks whether its spectral norm is below a prescribed threshold. If the check fails, the nonce is increased and a new candidate is sampled. This filtering procedure is mainly applied to the components of the secret vector, error vector, and encryption randomness, such as s, e, r, e_1 . The scalar error e_2 is sampled directly from the corresponding ternary distribution without additional filtering.

8.4 Computational Performance

Implementation on x86 PolarLAC has been implemented both in C language and with Intel AVX2 instructions for Intel x64 processors. In this section, we evaluate the performance of PolarLAC on Ubuntu 22.04 running on an Intel Core i9-11900K @ 3.50GHz processor with 32.0 GiB memory. Turbo Boost and Hyperthreading are disabled during the tests. The code is compiled with GCC 11.4.0, using the official compilation flags specified by the submission framework.

In the primary implementation, all hash and extendable-output functions are instantiated through the official SM3-based interface required by the submission framework. This SM3-based version is used as the default implementation for reporting the performance of PolarLAC. The corresponding cycle counts of the C and AVX2 implementations are listed in Tables 7 and 8.

For the SM3-based implementation, the AVX2 version reduces the total cost of PolarLAC.PKE.CPA by a factor of $1.08\times$ – $1.46\times$ compared with the C implementation. For PolarLAC.KEM.CCA, the

AVX2 implementation gives a speedup of up to $1.28\times$, while the overall speedup for the highest-security parameter set is less pronounced because the cost of SM3-based seed expansion and hash computations becomes a larger fraction of the total running time.

For reference, we also provide an alternative implementation in which the corresponding hash and XOF calls are instantiated with SHAKE. This SHAKE-based version keeps the same algorithmic structure, parameter sets, polynomial arithmetic routines, compression methods, and constant-time control flow as the SM3-based implementation; only the underlying hash/XOF primitive is replaced. Therefore, this version is intended only as a reference implementation for evaluating the impact of different symmetric-primitives on the overall cycle count, rather than as the default submission implementation. The corresponding cycle counts are reported in Tables 9 and 10.

In the SHAKE-based implementation, the AVX2 version achieves a larger reduction over its C counterpart. Specifically, the total cost of PolarLAC.PKE.CPA is reduced by a factor of $2.99\times$ – $4.69\times$, and the total cost of PolarLAC.KEM.CCA is reduced by a factor of $2.73\times$ – $4.08\times$. This indicates that, under the tested implementation setting, the performance contribution of the hash/XOF layer is non-negligible. We therefore report the SHAKE-based results only as an auxiliary comparison, while using the SM3-based implementation as the primary performance result.

Categories	C Implementation				AVX2 Implementation			
	KG	Enc	Dec	Total	KG	Enc	Dec	Total
PolarLAC-Light	132,743	156,827	41,700	331,270	104,100	107,466	16,050	227,617
PolarLAC-128	143,487	170,829	41,944	356,260	113,098	118,951	15,383	247,432
PolarLAC-256	267,550	351,464	90,555	709,569	206,208	301,174	35,280	542,662
PolarLAC-512	815,669	978,747	209,976	2,004,391	840,824	932,955	75,176	1,848,954
PolarLAC-512*	914,496	1,056,912	186,600	2,158,007	818,837	885,237	83,478	1,787,552

Table 7: Computational performance of the official SM3-based C and AVX2 implementations of PolarLAC.PKE.CPA.

Categories	C Implementation				AVX2 Implementation			
	KG	Encaps	Decaps	Total	KG	Encaps	Decaps	Total
PolarLAC-Light	142,480	189,902	239,763	572,145	117,979	154,714	179,777	452,470
PolarLAC-128	153,462	202,702	253,435	609,599	126,179	164,271	189,033	479,482
PolarLAC-256	277,609	383,564	495,614	1,156,787	220,977	311,580	369,797	902,355
PolarLAC-512	833,511	1,127,400	1,415,823	3,376,734	878,026	1,164,981	1,331,897	3,374,903
PolarLAC-512*	927,620	1,225,556	1,503,350	3,656,526	840,496	1,134,977	1,340,941	3,316,414

Table 8: Computational performance of the official SM3-based C and AVX2 implementations of PolarLAC.KEM.CCA.

Categories	C Implementation				AVX2 Implementation			
	KG	Enc	Dec	Total	KG	Enc	Dec	Total
PolarLAC-Light	90,819	114,226	41,705	246,750	17,739	21,209	16,312	55,260
PolarLAC-128	93,578	118,038	41,593	253,210	17,585	20,886	15,483	53,954
PolarLAC-256	188,753	240,395	90,662	519,810	38,019	48,373	35,546	121,938
PolarLAC-512	410,691	534,691	207,145	1,152,527	126,429	155,947	75,234	357,610
PolarLAC-512*	506,609	629,114	184,774	1,320,497	160,723	196,745	83,945	441,413

Table 9: Computational Performance of the SHAKE-based C and AVX2 Implementations of PolarLAC.PKE.CPA

Categories	C Implementation				AVX2 Implementation			
	KG	Encaps	Decaps	Total	KG	Encaps	Decaps	Total
PolarLAC-Light	93,915	123,769	173,877	391,561	18,463	30,509	55,823	104,794
PolarLAC-128	96,469	127,484	177,799	401,753	18,265	28,492	51,703	98,461
PolarLAC-256	192,452	256,353	363,959	812,764	39,135	64,938	119,196	223,270
PolarLAC-512	418,503	565,362	801,827	1,785,692	130,844	188,674	297,501	617,019
PolarLAC-512*	512,991	662,312	884,311	2,059,614	162,757	232,727	357,925	753,410

Table 10: Computational Performance of the SHAKE-based C and AVX2 Implementations of PolarLAC.KEM.CCA

Implementation on ARM We also implement PolarLAC on an ARMv8.2-A platform with SVE support to evaluate its portability and performance on modern ARM processors. The benchmark platform is a HiSilicon AArch64 server running Ubuntu 22.04 with GCC 11.4.0, and the measured SVE vector length is 256 bits. We report two build profiles in the ARM tables. The C baseline uses the finalized portable reference implementation and is compiled with the portable reference build using `-O2`, without `-march=armv8.2-a+sve`. The ARM/SVE implementation uses the optimized source files and fixed component switches described below, and is compiled with `-O3 -march=armv8.2-a+sve -fltto`.

The ARM implementation is derived from the finalized portable reference implementation. The optimized code is limited to implementation-level replacements of the same algorithms in the following components: polynomial arithmetic, including NTT/INTT, pointwise multiplication, and multiply-accumulate operations; sampling-layer helpers, including ternary sampling and integer-FFT spectral-bound helpers where enabled by the fixed configuration; and Polar decoding. The implementation does not change algorithm parameters, public-key or ciphertext formats, sampling distributions, rejection thresholds, or FO/KDF behavior.

The SM3/API-PKC route and the SHAKE route are built as two separate fixed compile-time configurations rather than by runtime backend switching. In the default SM3/API-PKC route, the polynomial-arithmetic component and decoder are enabled, the ternary sampler optimization is disabled, and the FFT-bound helper is enabled only for PolarLAC-512*. In the SHAKE route, the polynomial-arithmetic component, ternary sampler optimization, and decoder are enabled, and the FFT-bound helper is enabled for PolarLAC-Light, PolarLAC-128, PolarLAC-256, and PolarLAC-512, but disabled for PolarLAC-512*. In both routes, the uniform-matrix sampler remains part of the algorithm, but its dedicated ARM/SVE optimization component is not enabled; that sampler uses the reference implementation path compiled with the route’s optimization flags.

Tables 11 and 12 follow the same style as the x86 performance tables: they list the raw measurements for the portable C implementation and for the selected ARM/SVE implementation.

Category	C Implementation				ARM/SVE Implementation			
	KG	Encaps	Decaps	Total	KG	Encaps	Decaps	Total
PolarLAC-Light	54,518	70,599	87,780	212,897	51,061	64,789	77,491	193,342
PolarLAC-128	58,041	75,130	92,902	226,073	55,109	69,630	82,930	207,668
PolarLAC-256	108,299	145,856	185,056	439,211	96,490	128,970	156,870	382,331
PolarLAC-512	304,354	404,779	500,283	1,209,416	317,933	417,140	502,874	1,237,947
PolarLAC-512*	318,665	420,491	508,437	1,247,593	311,499	413,345	502,011	1,226,855

Table 11: Computational performance of the default SM3-based C and ARM/SVE implementations of PolarLAC.KEM.CCA (median ns/op).

Category	C Implementation				ARM/SVE Implementation			
	KG	Encaps	Decaps	Total	KG	Encaps	Decaps	Total
PolarLAC-Light	34,852	44,128	60,793	139,773	24,601	28,619	39,231	92,451
PolarLAC-128	35,709	45,339	62,147	143,195	19,897	24,125	34,756	78,779
PolarLAC-256	73,161	94,036	130,766	297,963	50,047	59,601	82,267	191,915
PolarLAC-512	147,711	190,911	266,777	605,399	100,157	121,150	170,719	392,026
PolarLAC-512*	162,544	207,270	272,194	642,008	98,168	119,596	168,609	386,373

Table 12: Computational performance of the SHAKE-based C and ARM/SVE implementations of PolarLAC.KEM.CCA (median ns/op).

Table 11 reports the default SM3/API-PKC ARM implementation. Compared with the portable C baseline, the selected ARM/SVE implementation gives end-to-end KEM speedups of 1.10x, 1.09x, 1.15x, 0.98x, and 1.02x for PolarLAC-Light, PolarLAC-128, PolarLAC-256, PolarLAC-512, and PolarLAC-512*, respectively. Thus, the default SM3 route gives modest end-to-end gains for most parameter sets and remains close to the portable C baseline for PolarLAC-512. Compiler flags alone were not sufficient on this platform: in separate compiler-only checks, the ARMv8.2-A+SVE build of the portable reference source was consistently slower than the portable -O2 baseline for the SM3 KEM measurements: the median total KEM latency was slower by 2.49%, 3.56%, 0.51%, 11.54%, and 9.86% for PolarLAC-Light, PolarLAC-128, PolarLAC-256, PolarLAC-512, and PolarLAC-512*, respectively. The selected ARM/SVE implementation uses hand-written arithmetic and decoder replacements to compensate for this regression and to recover modest end-to-end gains for most parameter sets.

Table 12 reports the SHAKE-based ARM implementation as an alternative instantiation. Compared with the portable C baseline, the selected SHAKE-based ARM/SVE implementation gives end-to-end KEM speedups of 1.51x, 1.82x, 1.55x, 1.54x, and 1.66x for PolarLAC-Light, PolarLAC-128, PolarLAC-256, PolarLAC-512, and PolarLAC-512*, respectively. The larger speedups in the SHAKE route reflect that the enabled ARM/SVE arithmetic, sampling, and decoder components account for a larger fraction of the total KEM latency in this instantiation.

9 Features

The proposed scheme PolarLAC is endowed with the following features.

Module LWE: The scheme is constructed over module lattices using the polynomial ring $\mathbb{Z}_q[x]/(x^n + 1)$ at all security levels. This ring is a broadly studied building block within a standard KEM framework equipped polar codes for error correction. Different security levels can be achieved by adjusting the corresponding ring parameters, modulus, and distributions. Consequently, optimized implementations can be easily adapted across different security levels.

NTT-Friendly Modulus: PolarLAC-128, -256, -512 chooses a NTT friendly moduli $q = 257$ to improve the efficiency. Despite 257 is larger than 1 byte, we optimize the sampling of the public key and storage of the ciphertext to ensure all polynomials are handled within 1 byte per coefficient. We select another NTT-friendly modulus $q = 769$ for PolarLAC-512* to ensure a DFR below 2^{-512} , for which efficient sampling and storage methods are also provided.

DFR Matched to the Security Level: To protect PolarLAC against DFR attacks, PolarLAC-128, -256, -512* are designed deliver DFR strictly matched to their target security levels. Experiments validate that even resource-exhaustion precomputation strategies fail to induce decryption failures, which proves that PolarLAC is immune to DFR-based attacks.

Spectral Rejection Sampling: To suppress the influence of the error correlations, we perform spectral rejection sampling to the secret and noise polynomials. This can effectively minimize the odds of extremely bad decryption errors under DFR attacks. Fundamentally, the proposed method efficiently filters out abnormal noise components while maintaining a minor rejection rate. In additional, we provide an alternative rejection sampling via autocorrelation polynomials which can achieve the same effectiveness as the spectral rejection sampling without FFT operations. This offers flexibility in selecting the suitable rejection sampling scheme according to the target architecture and implementation environment.

Polar Codes: PolarLAC is remarked by polar codes which leverage the “soft information” (i.e. likelihood ratio $\frac{Pr(\mathbf{y}|\mathbf{x}_1)}{Pr(\mathbf{y}|\mathbf{x}_0)}$) to recover the noisy plaintext message. This effectively increases the capability of error correction compared with the hard-decision decoding method. Meanwhile, polar codes allow code construction for arbitrary message lengths. This enables flexible bitwise adjustment of the code rate under a fixed code length. When the length of the message to be encrypted or the key to be negotiated is smaller than the target security level, one can switch to polar codes with the same set of parameters but a lower code rate to strengthen the robustness of the error correction module. Conversely, if the required message or negotiated key length exceeds the security level, one can increase the code rate to maintain a certain level of error correction capability within the system, as well as maintaining the CPA security.

References

1. Martin R. Albrecht. Lattice Estimator. <https://github.com/malb/lattice-estimator>.
2. Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2018.
3. Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In *Advances in Cryptology – ASIACRYPT 2020, Part II*, pages 583–613, 2020.
4. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. *IACR Cryptology ePrint Archive*, 2016:1157, 2016.
5. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343, 2016.
6. Erdal Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.
7. Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. Technical report, National Institute of Standards and Technology Post-Quantum Cryptography Standardization Project, 2021.

8. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, 2016.
9. Daniel J. Bernstein. Multi-ciphertext security degradation for lattices. *IACR Cryptol. ePrint Arch.*, 2022:1580, 2022.
10. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. *IACR Cryptology ePrint Archive*, 2017:634, 2017.
11. Dongshu Cai, Yijian Liu, Jiabo Wang, and Xianhui Lu. Thorns in polynomial convolution: Correlation, large deviations, and applications. *Cryptology ePrint Archive*, Paper 2026/1022, 2026.
12. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
13. Jan-Pieter D’Anvers and Senne Batsleer. Multitarget decryption failure attacks and their application to saber and kyber. In *Public-Key Cryptography – PKC 2022*, pages 3–33. Springer, 2022.
14. Jan-Pieter D’Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on ind-cca secure lattice-based schemes. In *Public-Key Cryptography – PKC 2019*, pages 565–598. Springer, 2019.
15. Jan-Pieter D’Anvers, Mélissa Rossi, and Fernando Virdia. (one) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.
16. Jan-Pieter D’Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. The impact of error dependencies on ring/mod-lwe/lwr based schemes. *IACR Cryptology ePrint Archive*, 2018:1172, 2018.
17. Jan-Pieter D’Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. *IACR Cryptology ePrint Archive*, 2018:1089, 2018.
18. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In *Advances in Cryptology - EUROCRYPT 2022, Part III*, pages 677–706, 2022.
19. Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC ’99, Kamakura, Japan, March 1-3, 1999, Proceedings*, pages 53–68, 1999.
20. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.
21. Lewis Glabush, Kathrin Hövelmanns, and Douglas Stebila. On the multi-target security of post-quantum key encapsulation mechanisms. *IACR Commun. Cryptol.*, 3(1):20, 2026.
22. Qian Guo, Thomas Johansson, and Jing Yang. A novel CCA attack using decryption errors against LAC. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 82–111. Springer, 2019.
23. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. *IACR Cryptology ePrint Archive*, 2017:604, 2017.
24. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.
25. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 96–125, 2018.
26. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75, 06 2014.
27. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In Jacques Stern, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers’ Track at the*

- RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
28. Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, and Zhenfei Zhang. Lac. Technical report, <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/LAC.zip>, 2017.
 29. Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, and Bao Li. Lac: Practical ring-lwe based public-key encryption with byte-level modulus. Technical report, <https://eprint.iacr.org/2018/1009>, 2018.
 30. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 35–54, 2013.
 31. MATZOV. Report on the security of LWE: Improved dual lattice attack. Technical report, Zenodo, 2022.
 32. National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report FIPS 203, U.S. Department of Commerce, 2024.
 33. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.
 34. Ido Tal and Alexander Vardy. How to construct polar codes. *IEEE Transactions on Information Theory*, 59(10):6562–6582, 2013.
 35. Ido Tal and Alexander Vardy. List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5):2213–2226, 2015.
 36. Anyu Wang, Zhongxiang Zheng, Chunhuan Zhao, Zhiyuan Qiu, Guang Zeng, Ye Yuan, Changchun Mu, and Xiaoyun Wang. Scloud+: An efficient lwe-based kem without ring/module structure. In *Security Standardisation Research: 9th International Conference, SSR 2024, Kunming, China, December 16, 2024, Proceedings*, page 147–174, Berlin, Heidelberg, 2024. Springer-Verlag.