LARK: Smaller and Faster Lattice-Based PRG

Achieving Sub-1 Cycle/Byte Performance

Yu Zhang[1,2], Jingnan He[1,2] (✉), Yijian Liu[1,2], Xianhui Lu[1,2] (✉), Dongshu Cai[1,2] and Kunpeng Wang[1,2]

[1] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China,
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{zhangyu1999,hejingnan,liuyijian,luxianhui,caidongshu,wangkunpeng}@iie.ac.cn

Abstract.
A key challenge in designing practical lattice-based pseudorandom number generators (PRGs) is that such generators typically produce byte-oriented output, whereas lattice parameters do not necessarily align with byte boundaries, creating a tradeoff between fully utilizing each computed value and maintaining efficient byte-oriented generation. To avoid additional packing overhead, LEAP (FSE 2025), a state-of-the-art lattice-based PRG built on Ring Learning with Rounding (RLWR), retains only the lowest eight bits of each RLWR value for byte-oriented generation and discards the remaining high-order bits. Consequently, this high-order information contributes neither to LEAP's output nor its security. We present LARK, a lattice-based pseudorandom generator that directly exposes the low-order byte as pseudorandom output and uses the discarded high-order information as security-relevant hidden uncertainty. This additional uncertainty allows LARK to meet its target security levels with smaller parameters, yielding compact keys and efficient implementations.
To justify using the discarded high-order information as hidden uncertainty, LARK replaces LEAP's RLWR-based output-generation component with one based on Ring Learning with Alternating Moduli (RLAM). For efficient polynomial arithmetic, we formulate RLAM as a ring analogue of the LAM problem introduced by Chen et al. (CRYPTO 2026), in which LWR rounding is replaced by direct reduction between coprime moduli. This modulus reduction introduces statistical bias, which LARK removes with lightweight masked rejection. We prove that LARK is pseudorandom under the decisional NTRU and RLAM assumptions. Compared with LEAP, LARK reduces key sizes by 42.85% and 68.00% at the 128-bit and 256-bit security levels, respectively, while consistently improving performance. LARK improves performance by up to 37.16% in plain C, 10.55% with AVX2, and 21.05% with AVX512. In particular, the AVX512 implementations achieve 0.90 and 0.91 cycles per byte at the two security levels, attaining sub-1 cycle/byte performance at both levels.

Keywords: Lattice-based cryptography · PRG · NTRU · LAM

1 Introduction

Pseudorandom number generators (PRGs) are fundamental cryptographic primitives that expand a short uniformly random seed into a longer sequence that is computationally indistinguishable from a uniform distribution. They play a central role in modern cryptographic systems, where they are used to derive keys from compact seeds, generate nonces and masks in symmetric encryption and authentication schemes, and supply randomness to higher-level protocols. In practice, pseudorandom generation is commonly realized through symmetric constructions, ranging from stateful generators such as Yarrow [KSF99]

Licensed under Creative Commons License CC-BY 4.0.

and Fortuna [FSK15] to block-cipher-based generators such as AES-CTR [BK15] and Keccak-based extendable-output functions such as SHAKE [oSND15] and TurboSHAKE [BDH+23]. These designs have benefited from extensive cryptanalysis, decades of implementation optimization, and dedicated hardware support. Their compact states and excellent throughput make them highly attractive for real-world systems and establish the practical benchmark for alternative PRG designs.

In addition to PRGs built from symmetric primitives, there exist constructions whose security is based on well-studied computational hardness assumptions. Such constructions provide a complementary route to pseudorandomness by tying their security to an explicit computational problem. More recently, lattice-based hardness assumptions have been explored as a foundation for constructing symmetric primitives, including pseudorandom number generators. In particular, the Learning With Rounding (LWR) problem and its ring variant, Ring Learning With Rounding (RLWR), are well suited to deterministic symmetric constructions, since they replace the explicit random error in Learning With Errors (LWE) and Ring Learning With Errors (RLWE) with deterministic rounding while retaining hardness under suitable parameters, with outputs of the form $\lfloor (p/q) \cdot (\langle \mathbf{a}, \mathbf{s} \rangle \bmod q) \rceil \bmod p$. The algebraic structure of the ring variants further enables efficient polynomial arithmetic using techniques such as the Number Theoretic Transform (NTT). Banerjee, Peikert, and Rosen [BPR12] introduced LWR and used it to construct the first direct lattice-based pseudorandom functions. While their reductions provided a strong theoretical foundation, the resulting parameters were too large for efficient practical instantiation. Subsequent work therefore focused on more practical parameter choices supported by concrete cryptanalysis. The SPRING family [BBL+15] introduced efficient rounded-ring instantiations and used NTT-friendly moduli to accelerate polynomial arithmetic, together with dedicated techniques to correct the bias introduced by rounding. Building on this line, SPRING-RS [BDFK17] replaced the more involved bias-correction mechanisms with rejection sampling, yielding a simpler and more efficient lattice-based pseudorandom generator. Most recently, LEAP [ZLL+25] introduced new techniques for parameter reuse and key compression, achieving significant improvements in both key size and throughput compared with earlier SPRING-based generators. To date, it represents the best-performing lattice-based PRG. Nevertheless, compared with the fastest generators built from conventional symmetric primitives, lattice-based designs still incur non-negligible overhead in both key size and performance.

Further narrowing this gap requires not only optimizing the underlying computation but also making effective use of each arithmetic result. Lattice parameters are selected jointly to satisfy security and computational requirements, and the resulting values do not necessarily align with byte boundaries. In byte-oriented implementations, complete bytes can be handled efficiently, whereas using the remaining bits requires additional packing and processing. This creates a tradeoff between making fuller use of each computed value and maintaining efficient byte-oriented generation.

This tradeoff appears concretely in LEAP, where the output of the RLWR computation serves as a source of pseudorandom bytes. In its practical parameter sets, the moduli q and p lie between 8 and 16 bits, providing a favorable balance between security and implementation efficiency. Since pseudorandom outputs are typically stored as byte arrays, LEAP extracts only the lowest 8 bits, namely one byte, from each rounded value to simplify byte alignment and avoid additional packing overhead. The remaining higher-order bits, which amount to less than one byte, are therefore discarded. This choice is not due to a formal restriction that prevents setting $p = 256$ in an RLWR-based design. Instead, LEAP selects its parameters by jointly optimizing RLWR security, byte extraction, rejection efficiency, and fast shift-based rounding, and directly setting $p = 256$ does not provide the best overall tradeoff in this parameter regime. Thus, even in the best-performing lattice-based PRG, part of the information produced by the underlying computation is left

unused. This naturally raises the question of whether the discarded information can be utilized more effectively without sacrificing efficient byte-oriented generation.

1.1 Our Contributions

In this work, we introduce LARK, a lattice-based pseudorandom number generator with compact keys and high performance. At a high level, LARK is motivated by a different way of utilizing the high-order information discarded in LEAP for implementation efficiency. Rather than recovering this information as additional output, we incorporate it into the concrete security estimation as security-relevant uncertainty. The resulting efficiency gain arises indirectly from this revised security accounting. For fixed lattice parameters, the additional hidden information increases the estimated cost of attacking the underlying problem, providing extra security margin. LARK uses this margin to select smaller parameters while preserving the target 128-bit and 256-bit security levels. These smaller parameters reduce both the key size and the cost of polynomial arithmetic. Realizing this design requires a different hard problem for output generation, an efficient mechanism for eliminating the statistical bias introduced by the new assumption, and a complete integration of these components into a secure PRG.

Hardness foundation To formalize this information allocation, we require a hard problem in which the low-order portion can be exposed as output while the high-order portion remains hidden. This perspective is closely related to Learning with Rounding (LWR) through the well-known correspondence between least-significant-bit and most-significant-bit encodings. In particular, Alperin-Sheriff and Peikert [ASP13a] showed that, when p and q are coprime, these two encodings in (Ring-)LWE-based cryptosystems can be converted into each other by multiplication by p or p^{-1} modulo q, together with the corresponding adjustment of the encoded message modulo p. This establishes a close algebraic relationship between modulus reduction and rounding at the encoding level, but does not by itself establish equivalence between the corresponding formal hard problems. Establishing such an equivalence would further require rigorous reductions that account for the choice of integer representatives and rounding conventions, and demonstrate the appropriate correspondence between the challenge and reference distributions in the respective distinguishing experiments. Chen et al. [CJL26] subsequently formalized the Learning with Alternating Moduli (LAM) problem, whose output is of the form $(\langle \mathbf{a}, \mathbf{s} \rangle \bmod q) \bmod p$, and investigated its hardness under suitable parameter settings. To support efficient computation in the polynomial-ring setting, we introduce the Ring Learning with Alternating Moduli (RLAM) problem as its natural ring analogue and establish its hardness via a reduction from Ring Learning with Errors (RLWE). The key advantage of the RLAM viewpoint becomes particularly apparent in the symmetric setting. In public-key encryption schemes based on LWE or LWR, noise must be carefully controlled to ensure correct decryption, which constrains both its magnitude and its placement within ciphertext components. In particular, the effective noise is typically confined to the lower-order portion, since perturbations affecting higher-order bits may translate into larger decoding errors and thereby increase the probability of decryption failure. Symmetric constructions such as pseudorandom number generators, by contrast, involve no decryption procedure and therefore impose no analogous correctness constraint on noise growth. Consequently, as long as the resulting output distribution remains computationally indistinguishable from uniform, the position of the exposed output within the bit representation is no longer dictated by decryption correctness. This additional flexibility makes the RLAM formulation particularly well suited to efficient lattice-based pseudorandom generation.

Exact output distribution Although RLAM, unlike RLWR, requires the moduli q and p to be coprime, this requirement introduces essentially no performance penalty in our practical setting. Indeed, LEAP, which is based on RLWR, also adopts prime moduli in its parameter sets, since polynomial multiplication dominates the overall computational cost and the moduli are therefore chosen primarily to support efficient Number Theoretic Transform (NTT) implementations. More importantly, the RLAM-based formulation allows us to select suitable parameters with $p = 256$. This choice aligns the output modulus directly with one byte, so the low-order 8 bits can be exposed as pseudorandom output without additional packing, while all remaining high-order information can be retained as security-relevant uncertainty rather than being discarded. The use of coprime moduli, however, introduces another issue: directly reducing a uniformly distributed element of $\mathbb{Z}_q$ modulo p does not, in general, yield a uniform element of $\mathbb{Z}_p$, since the q possible input values cannot be evenly distributed among the p residue classes. To eliminate this bias, we introduce a lightweight masked-rejection technique. Specifically, we choose the moduli such that $q = zp + 1$. Under this condition, the bias is concentrated entirely at the residue 0: the value 0 has $z + 1$ preimages in $\mathbb{Z}_q$, whereas every nonzero residue has exactly z preimages. Consequently, exact uniformity over $\mathbb{Z}_p$ can be restored by always accepting nonzero coefficients and rejecting each zero coefficient with probability $1/(z + 1)$. We further choose $z + 1 = 2^v$, allowing the rejection decision to be implemented efficiently using a uniformly sampled v-bit mask. For each zero coefficient, the corresponding mask value determines whether it is accepted, whereas nonzero coefficients are always accepted. This carefully chosen modulus relation therefore reduces the uniformization procedure to inexpensive masking and comparison operations, making masked rejection particularly efficient.

Construction and security LARK's structural design builds on the framework introduced in LEAP [ZLL+25]. More precisely, LARK inherits from LEAP the idea of treating the public parameter as part of the key in the PRG setting, the two-phase GEN/PRG framework for key-size compression, and the reuse framework that simultaneously reuses a and s. Within this framework, LARK replaces the RLWR-based output-generation component with an RLAM-based formulation and incorporates masked rejection to ensure an exactly uniform output distribution. The more compact parameter regime also allows the GEN phase to rely on the standard NTRU assumption rather than the vectorized-NTRU assumption used by LEAP, further reducing the key size while grounding the construction in a more extensively studied hardness assumption with a broader body of theoretical and cryptanalytic work. We prove that the resulting construction is pseudorandom under the decisional NTRU and RLAM assumptions.

From hidden uncertainty to efficiency We next clarify how this additional hidden uncertainty translates into practical improvements. In the concrete security estimation of RLAM, the high-order information discarded in LEAP contributes to the effective error distribution. Accounting for this additional error leads to a higher estimated attack cost for fixed parameters, providing additional security margin in the concrete security evaluation. LARK uses this margin to select smaller parameters while preserving the target 128-bit and 256-bit security levels. In particular, LARK uses polynomial dimension $n = 128$ and modulus $q = 769$, compared with $n = 256$ and $q = 3329$ in the LEAP. These parameter reductions improve key size and performance through distinct but complementary mechanisms. The more compact parameter regime allows the GEN phase to rely on standard NTRU rather than the vectorized-NTRU formulation used by LEAP, reducing the key sizes to 128 and 256 bytes at the two security levels. The smaller polynomial dimension also provides direct implementation benefits, enabling fully complete NTT implementations with substantially reduced memory traffic. This advantage is

particularly pronounced with AVX512. Under our parameter choices, an entire polynomial can remain in SIMD registers throughout the core NTT computation, while AVX512 allows four polynomials to be processed in parallel, improving vector utilization and instruction-level parallelism. As a result, our AVX512 implementation achieves 0.90 cycles/byte at the 128-bit security level and 0.91 cycles/byte at the 256-bit security level, attaining sub-1 cycle/byte performance at both security levels. Compared with AES-CTR at matching security levels, the key-size ratio decreases from 14–25× for LEAP to 8× for LARK, while the plain-C cycles/byte overhead decreases from approximately 67–124% to 7–40%.

1.2 Comparison to the State of the Art

In Table 1, we provide a comprehensive comparison of LARK with LEAP, the SPRING family, AES-CTR, and Keccak in terms of key sizes, public parameter sizes, and the number of CPU cycles required to generate one byte on an identical machine. AES-CTR is one of the fastest and most widely deployed pseudorandom number generators. We evaluate a highly optimized AES-CTR implementation from the SUPERCOP-20260217 [1] benchmark suite, which incorporates one of the fastest known AES implementations across different security levels. The reported results include both AES-NI–accelerated implementations and plain C implementations without AES-NI support. To provide a stronger hardware-accelerated baseline, we additionally evaluate AES-CTR implementations accelerated by 512-bit VAES instructions from Intel's Multi-Buffer Crypto for IPsec Library[2] at both the 128-bit and 256-bit security levels. In addition, we report performance results for SHAKE and TurboSHAKE[3]. SHAKE128 and SHAKE256 are FIPS 202 extendable-output functions based on the Keccak sponge construction, while TurboSHAKE128 and TurboSHAKE256 are reduced-round Keccak-based XOFs designed for higher throughput. We benchmark the raw single-stream XOF implementations in plain C, AVX2, and AVX512. Since the optimized 64-bit C implementation is already efficient for single-stream Keccak, AVX2 provides only a modest gain, while AVX512 achieves a larger improvement by accelerating the underlying Keccak-p permutation.

Table 1: Comparison between LEAP, SPRING family, AES-CTR and SHAKE

Scheme	Key (Bytes)	Public (Bytes)	Cycles/Byte					Security Level
			C	AVX2	AVX512	AES-NI	VAES	
LARK-128	128	-	8.30	1.44	0.90	-	-	128-bit
LEAP-128	224	-	13.21	1.61	1.14	-	-	
SPRING-RS	385	-	-	5.32	-	-	-	
SPRING-CRT	18444	145	-	20.85	-	-	-	
SPRING-BCH	8198	129	-	40.65	-	-	-	
AES128-CTR	16	-	5.91	-	-	0.33	0.16	
SHAKE128	16	-	8.90	7.51	4.49	-	-	
TurboSHAKE128	16	-	4.35	4.15	2.26	-	-	
LARK-256	256	-	8.64	1.51	0.91	-	-	256-bit
LEAP-256	800	-	13.42	1.61	1.14	-	-	
AES256-CTR	32	-	8.05	-	-	0.45	0.22	
SHAKE256	32	-	11.00	9.26	5.55	-	-	
TurboSHAKE256	32	-	5.52	5.13	2.79	-	-	

For LARK, we present performance results at the 128-bit and 256-bit security levels, denoted LARK-128 and LARK-256, respectively, implemented in plain C, AVX2, and AVX512. Compared with LEAP, the current best-performing lattice-based PRG, LARK reduces the key size by 42.85% at the 128-bit security level and by 68.00% at the 256-bit

[1] https://bench.cr.yp.to/supercop.html
[2] https://github.com/intel/intel-ipsec-mb, commit 5fd2d10fa27f
[3] https://github.com/XKCP/XKCP

security level. Beyond these substantial key-size reductions, LARK also delivers consistent performance improvements, achieving gains of up to 37.16%, 10.55%, and 21.05% in the plain C, AVX2, and AVX512 implementations, respectively. Notably, the AVX512 implementation attains 0.90 cycles/byte for LARK-128 and 0.91 cycles/byte for LARK-256, achieving sub-1 cycle/byte performance at both security levels.

1.3 Related Work

A separate but related line of research studies pseudorandom constructions that alternate between different moduli. Boneh, Ishai, Passelègue, Sahai, and Wu [BIP+18] initiated this direction by proposing weak-PRF candidates that compose linear operations over small moduli, most notably modulo 2 and modulo 3. These constructions achieve very low circuit complexity and are particularly attractive for MPC, while their security is primarily supported by conjectures and resistance to known attacks rather than reductions to established hardness assumptions. Subsequent work further examined the security of this paradigm. Hu et al. [HLR+26] developed new cryptanalytic attacks against two alternating-moduli weak-PRF constructions and showed that some previously proposed parameter sets do not achieve their targeted security levels. Chen et al. [CJL26] later formalized the underlying modular structure through the Learning with Alternating Moduli (LAM) problem. For suitable parameters, they established the hardness of LAM from LWE and constructed low-depth weak-PRF candidates based on LAM or LWR, providing a more explicit hardness foundation for alternating-moduli constructions. Chen et al. and LARK use the alternating-moduli framework for different purposes. Chen et al. employ LAM to construct low-depth weak PRFs whose pseudorandomness is required only on uniformly random inputs, whereas LARK introduces the ring analogue of LAM to support a practical byte-oriented PRG with efficient polynomial arithmetic.

2 Preliminaries

In this section, we first introduce the necessary mathematical notation, then define the lattice-based hard problems and pseudorandom number generators, and finally describe the Number Theoretic Transform.

2.1 Notation

Let $\mathbb{Z}_q$ denote the residue class ring modulo an integer $q \geq 1$. Let $\mathcal{R}_n = \mathbb{Z}[x]/(x^n + 1)$ and $K_{\mathbb{R}} = \mathbb{R}[x]/(x^n + 1)$, where n is a power of two. We write $\mathcal{R}_{q,n} = \mathcal{R}_n/q\mathcal{R}_n$ for the corresponding quotient ring, and denote by $\mathcal{R}^*_{q,n}$ its group of units. We further define $\mathcal{R}^\dagger_{q,n} := \{x \in \mathcal{R}_{q,n} \mid \gcd(q, x_0, x_1, \cdots, x_{n-1}) = 1\}$.

Regular-font letters denote elements of $\mathcal{R}_n$ or $\mathcal{R}_{q,n}$, including elements of $\mathbb{Z}$ and $\mathbb{Z}_q$. Bold lowercase letters represent vectors with entries in $\mathbb{Z}$ or $\mathbb{Z}_q$, or in $\mathcal{R}_n$ or $\mathcal{R}_{q,n}$. We denote by $\mathbf{B}^l$ the set of byte arrays of length l, and by $\mathbf{b}^l \in \mathbf{B}^l$ a byte array of length l. By default, all vectors are column vectors. Bold uppercase letters denote matrices. For a ring element a, we write a_i to denote the coefficient of x^i. For a vector $\mathbf{v}$, $\mathbf{v}_i$ denotes its i-th entry, and for a matrix $\mathbf{A}$, $\mathbf{A}_{i,j}$ denotes the entry in the i-th row and j-th column. We write $\mathbf{b}_{i:j}$ to denote the subarray of $\mathbf{b}$ consisting of elements with indices from i to j. All indices are zero-based.

We define a rounding function $\lfloor\cdot\rceil_{q\to p} : \mathbb{Z}_q \to \mathbb{Z}_p$ for integers $q \geq p \geq 2$ as $\lfloor x\rceil_{q\to p} = \lfloor (p/q) \cdot \bar{x}\rceil \bmod p$, where $\bar{x} \in \mathbb{Z}$ is any integer congruent to $x \bmod q$. We also define a modulus-reduction function $[\cdot]_{q\to p} : \mathbb{Z}_q \to \mathbb{Z}_p$ for integers $q \geq p \geq 2$ as $[x]_{q\to p} := (\bar{x} \bmod q) \bmod p$, where $\bar{x} \in \mathbb{Z}$ is any integer representative of the residue class $x \in \mathbb{Z}_q$, and $\bar{x} \bmod q$ is always taken as the unique representative in $\{0, 1, \ldots, q-1\}$. Both the

rounding function and the modulus reduction function are applied component-wise to vectors and matrices and coefficient-wise to elements of the ring.

For a set S, $U(S)$ denotes the uniform distribution over S. For a distribution $\mathcal{X}$, the notation $x \leftarrow \mathcal{X}$ means that x is sampled according to $\mathcal{X}$.

2.2 Hardness Assumption

2.2.1 NTRU

The NTRU problem was introduced by Hoffstein, Pipher, and Silverman [HPS98]. Notably, in power-of-two cyclotomic rings, it was shown in [SS11] that when the distribution $\mathcal{X}$ is a discrete Gaussian with standard deviation $\sigma = \tilde{w}(n\sqrt{q})$, the resulting distribution is statistically indistinguishable from uniform, thereby providing a firm theoretical foundation. Moreover, the NTRU problem has been extensively studied and adopted in subsequent works [DLP14, FHK+19, PMS21], further establishing its robustness and theoretical soundness.

Definition 1. Let $n \geq 1$ be the main security parameter and let $q \geq 2$ be an integer.

- Let $\mathcal{X}_g$ be a distribution over $\mathcal{R}_{q,n}$, and let $\mathcal{X}_f$ be a distribution over $\mathcal{R}^*_{q,n}$. The decisional NTRU problem $\mathrm{DNTRU}_{n,q,\mathcal{X}_g,\mathcal{X}_f}$ is to distinguish between the distributions g/f and u, where $g \leftarrow \mathcal{X}_g$, $f \leftarrow \mathcal{X}_f$ and $u \leftarrow U(\mathcal{R}_{q,n})$.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional NTRU problem $\mathrm{DNTRU}_{n,q,\mathcal{X}_g,\mathcal{X}_f}$ is defined as

$$\mathrm{Adv}^{\mathrm{DNTRU}}_{n,q,\mathcal{X}_g,\mathcal{X}_f}(\mathcal{A}) = |\Pr[\mathcal{A}(g/f) = 1] - \Pr[\mathcal{A}(u) = 1]|$$

The NTRU assumption states that for appropriate parameters (n, q) and distribution $\mathcal{X}$, no probabilistic polynomial-time adversary can solve $\mathrm{DNTRU}_{n,q,\mathcal{X}_g,\mathcal{X}_f}$ with non-negligible advantage.

2.2.2 Learning With Errors

Most lattice-based cryptographic schemes are currently designed based on the Learning With Errors (LWE) problem, initially introduced by Regev [Reg05]. Subsequently, Lyubashevsky, Peikert, Regev [LPR10], and Stehlé et al. [SSTX09] introduced a ring-based version of LWE, referred to as Ring Learning With Errors (RLWE), and demonstrated that the security of RLWE can be reduced to the worst-case problems on ideal lattices. Further research [PRSD17] has shown that by selecting an appropriate error distribution, the RLWE problem is hard for any ring, which provides solid theoretical support for the application of RLWE.

Definition 2. Let $n \geq 1$ be the main security parameter and let $q \geq 2$ be an integer.

- Let $\mathcal{X}_{\mathbf{s}}$ be a distribution over $\mathbb{Z}_q^n$ and let $\mathcal{X}_e$ be a distribution over $\mathbb{Z}_q$. For a secret $\mathbf{s} \leftarrow \mathcal{X}_{\mathbf{s}}$, the decisional LWE problem, denoted by $\mathrm{DLWE}_{m,n,q,\mathcal{X}_{\mathbf{s}},\mathcal{X}_e}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \bmod q) \quad \text{and} \quad (\mathbf{a}_i, u_i),$$

for $i \in \{0, \ldots, m-1\}$, where $\mathbf{a}_i \leftarrow U(\mathbb{Z}_q^n)$, $e_i \leftarrow \mathcal{X}_e$, and $u_i \leftarrow U(\mathbb{Z}_q)$.

- Let $\mathcal{X}_s$ and $\mathcal{X}_e$ be distributions over $\mathcal{R}_{q,n}$. For a secret $s \leftarrow \mathcal{X}_s$, the decisional RLWE problem, denoted by $\mathrm{DRLWE}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(a_i, b_i = a_i s + e_i) \quad \text{and} \quad (a_i, u_i),$$

for $i \in \{0, \ldots, m-1\}$, where $a_i \leftarrow U(\mathcal{R}_{q,n})$, $e_i \leftarrow \mathcal{X}_e$ and $u_i \leftarrow U(\mathcal{R}_{q,n})$.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional LWE problem $\mathrm{DLWE}_{m,n,q,\mathcal{X}_\mathbf{s},\mathcal{X}_e}$ is defined as

$$\mathrm{Adv}^{\mathrm{DLWE}}_{m,n,q,\mathcal{X}_\mathbf{s},\mathcal{X}_e}(\mathcal{A}) = |\Pr[\mathcal{A}((\mathbf{a}_0, b_0), \ldots, (\mathbf{a}_{m-1}, b_{m-1})) = 1]$$

$$- \Pr[\mathcal{A}((\mathbf{a}_0, u_0), \ldots, (\mathbf{a}_{m-1}, u_{m-1})) = 1]|.$$

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional RLWE problem $\mathrm{DRLWE}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}$ is defined as

$$\mathrm{Adv}^{\mathrm{DRLWE}}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}(\mathcal{A}) = |\Pr[\mathcal{A}((a_0, b_0), \ldots, (a_{m-1}, b_{m-1})) = 1]$$
$$- \Pr[\mathcal{A}((a_0, u_0), \ldots, (a_{m-1}, u_{m-1})) = 1]|.$$

The LWE assumption states that for appropriate parameters (m, n, q), secret distribution $\mathcal{X}_\mathbf{s}$ and error distribution $\mathcal{X}_e$, no probabilistic polynomial-time adversary can solve $\mathrm{DLWE}_{m,n,q,\mathcal{X}_\mathbf{s},\mathcal{X}_e}$ with non-negligible advantage. The RLWE assumption is defined analogously.

2.2.3 Learning with Rounding

To generate error terms in LWE efficiently and deterministically while preserving hardness, Banerjee, Peikert, and Rosen [BPR12] introduced the Learning With Rounding (LWR) problem and its ring-based variant, Ring Learning With Rounding (RLWR). The core idea of this derandomization technique is to replace the addition of a small random error term by deterministic rounding of the inner product $\langle \mathbf{a}, \mathbf{s} \rangle$ to a nearby value. They showed that when the modulus q is exponentially larger than the rounding modulus p, the LWR problem can be reduced to the LWE problem. Moreover, this reduction remains valid for polynomially sized moduli q, provided that the number of available LWR samples is appropriately bounded [AKPW13, BGM+16, ASA16, LW20].

Definition 3. Let $n \geq 1$ be the main security parameter and let $q \geq p \geq 2$ be integers.

- Let $\mathcal{X}_\mathbf{s}$ be a distribution over $\mathbb{Z}_q^n$. For a secret $\mathbf{s} \leftarrow \mathcal{X}_\mathbf{s}$, the decisional LWR problem, denoted by $\mathrm{DLWR}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(\mathbf{a}_i, b_i = \lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_{q \to p}) \quad \text{and} \quad (\mathbf{a}_i, \lfloor u_i \rceil_{q \to p}),$$

for $i \in \{0, \ldots, m-1\}$, where $\mathbf{a}_i \leftarrow U(\mathbb{Z}_q^n)$ and $u_i \leftarrow U(\mathbb{Z}_q)$.

- Let $\mathcal{X}_s$ be a distribution over $\mathcal{R}_{q,n}$. For a secret $s \leftarrow \mathcal{X}_s$, the decisional RLWR problem, denoted by $\mathrm{DRLWR}_{m,n,q,p,\mathcal{X}_s}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(a_i, b_i = \lfloor a_i \cdot s \rceil_{q \to p}) \quad \text{and} \quad (a_i, \lfloor u_i \rceil_{q \to p}),$$

for $i \in \{0, \ldots, m-1\}$, where $a_i \leftarrow U(\mathcal{R}_{q,n})$ and $u_i \leftarrow U(\mathcal{R}_{q,n})$.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional LWR problem $\mathrm{DLWR}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$ is defined as

$$\mathrm{Adv}^{\mathrm{DLWR}}_{m,n,q,p,\mathcal{X}_\mathbf{s}}(\mathcal{A}) = |\Pr[\mathcal{A}((\mathbf{a}_0, b_0), \ldots, (\mathbf{a}_{m-1}, b_{m-1})) = 1]$$

$$- \Pr[\mathcal{A}((\mathbf{a}_0, \lfloor u_0 \rceil_{q\to p}), \ldots, (\mathbf{a}_{m-1}, \lfloor u_{m-1} \rceil_{q\to p})) = 1]|.$$

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional RLWR problem $\mathrm{DRLWR}_{m,n,q,p,\mathcal{X}_s}$ is defined as

$$\mathrm{Adv}^{\mathrm{DRLWR}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{A}) = |\Pr[\mathcal{A}((a_0, b_0), \ldots, (a_{m-1}, b_{m-1})) = 1]$$
$$- \Pr[\mathcal{A}((a_0, \lfloor u_0 \rceil_{q\to p}), \ldots, (a_{m-1}, \lfloor u_{m-1} \rceil_{q\to p})) = 1]|.$$

The LWR assumption states that for appropriate parameters (m, n, q, p) and secret distribution $\mathcal{X}_\mathbf{s}$, no probabilistic polynomial-time adversary can solve $\mathrm{DLWR}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$ with non-negligible advantage. The RLWR assumption is defined analogously.

2.2.4 Learning with Alternating Moduli

The LAM problem can be viewed as a deterministic variant of LWE in which the random error term is replaced by a reduction across two coprime moduli. Alperin-Sheriff and Peikert [ASP13b] observed a close connection between modular reduction and rounding, showing that two (Ring-)LWE encodings based on these operations can be efficiently converted into each other when the corresponding moduli are coprime. This observation reveals a close relation between LAM-like modular reduction and LWR-style rounding. Chen et al. [CJL26] formalized this modular-reduction paradigm as the Learning with Alternating Moduli (LAM) problem and showed that, under appropriate parameter settings, LAM is at least as hard as LWE.

Definition 4. Let $n \geq 1$ be the main security parameter and let $q > p \geq 2$ be coprime integers.

- Let $\mathcal{X}_\mathbf{s}$ be a distribution over $\mathbb{Z}_q^n$. For a secret $\mathbf{s} \leftarrow \mathcal{X}_\mathbf{s}$, the decisional LAM problem, denoted by $\mathrm{DLAM}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(\mathbf{a}_i, b_i = [\langle \mathbf{a}_i, \mathbf{s}\rangle]_{q\to p}) \quad \text{and} \quad (\mathbf{a}_i, [u_i]_{q\to p}),$$

  for $i \in \{0, \ldots, m-1\}$, where $\mathbf{a}_i \leftarrow U(\mathbb{Z}_q^n)$ and $u_i \leftarrow U(\mathbb{Z}_q)$.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional LAM problem $\mathrm{DLAM}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$ is defined as

$$\mathrm{Adv}^{\mathrm{DLAM}}_{m,n,q,p,\mathcal{X}_\mathbf{s}}(\mathcal{A}) = |\Pr[\mathcal{A}((\mathbf{a}_0, b_0), \ldots, (\mathbf{a}_{m-1}, b_{m-1})) = 1]$$

$$- \Pr[\mathcal{A}((\mathbf{a}_0, [u_0]_{q\to p}), \ldots, (\mathbf{a}_{m-1}, [u_{m-1}]_{q\to p})) = 1]|.$$

The LAM assumption states that for appropriate parameters (m, n, q, p) and secret distribution $\mathcal{X}_\mathbf{s}$, no probabilistic polynomial-time adversary can solve $\mathrm{DLAM}_{m,n,q,p,\mathcal{X}_\mathbf{s}}$ with non-negligible advantage.

2.3 Pseudorandom Number Generators

The concept of pseudorandomness was formalized by Blum and Micali [BM82] and by Yao [Yao82], who showed that pseudorandom sequences can safely replace truly random ones in any polynomial-time algorithm. Pseudorandom number generators significantly reduce the need for true randomness: a single random seed sampled during setup suffices to deterministically generate all required randomness throughout execution. In particular, any cryptographic construction that remains secure when using truly random bits also remains secure when these bits are replaced by pseudorandom outputs of a secure PRG [Yao82].

Definition 5. Let $l(\cdot) : \mathbb{N} \to \mathbb{N}$ be a function such that $l(n) > n$ for all $n \in \mathbb{N}$. A pseudorandom number generator with stretch function l is a deterministic polynomial-time algorithm G that satisfies:

- **Expansion:** $|G(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.
- **Pseudorandom:** $\{G(\{0,1\}^n)\}_{n\in\mathbb{N}}$ and $\{\{0,1\}^{l(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable distributions.

2.4 Number Theoretic Transform

For polynomial multiplication over finite rings, classical schoolbook multiplication has quadratic complexity $O(n^2)$. To achieve higher performance, lattice-based cryptographic schemes typically rely on techniques derived from the Fast Fourier Transform (FFT) [CT65] and its finite-field analogue, the Number Theoretic Transform (NTT) [Pol71]. These methods map polynomials from the coefficient domain into a transformed representation in which multiplication becomes component-wise, thereby reducing the overall complexity to $O(n \log n)$.

From an algebraic perspective, the NTT exploits ring isomorphisms induced by factorizations of the modulus polynomial. Let $\mathcal{R}_q = \mathbb{Z}_q[x]/f(x)$. If $f(x)$ factors as $f(x) = g(x) \cdot h(x)$ over $\mathbb{Z}_q$ with $\gcd(g, h) = 1$, then the Chinese Remainder Theorem yields the isomorphism

$$\mathbb{Z}_q[x]/f(x) \cong \mathbb{Z}_q[x]/g(x) \times \mathbb{Z}_q[x]/h(x).$$

By recursively applying such factorizations, polynomial arithmetic in $\mathcal{R}_q$ can be decomposed into lower-dimensional subproblems that can be computed independently and recombined via the inverse isomorphism.

Complete NTT. Consider $\mathcal{R}_{q,n} = \mathbb{Z}_q[x]/(x^n + 1)$ with n a power of two. If $\mathbb{Z}_q$ contains a primitive $2n$-th root of unity ζ, then $x^n + 1$ factors completely into linear terms, yielding an isomorphism to a direct product of n degree-one polynomials. In this setting, which is referred to as the complete NTT, polynomial multiplication reduces entirely to componentwise multiplication.

Incomplete NTT. If a primitive $2n$-th root of unity does not exist in $\mathbb{Z}_q$, one may instead use a primitive $2n/k$-th root of unity for some divisor $k \mid n$. In this case, $x^n + 1$ decomposes into n/k factors of degree k, resulting in an incomplete NTT. Polynomial multiplication proceeds analogously to the complete case, except that the lowest-level operations involve $k \times k$ convolutions. Despite this, the incomplete NTT still provides substantial efficiency gains and greater flexibility in parameter selection.

Using the NTT, the multiplication of two polynomials $a, b \in \mathcal{R}_{q,n}$ can be computed as

$$a \cdot b = \mathbf{INTT}_q^n\big(\mathbf{BaseMul}_q^n\big(\mathbf{NTT}_q^n(a), \mathbf{NTT}_q^n(b)\big)\big)\,,$$

where $\mathbf{BaseMul}_q^n$ performs pointwise multiplication for the complete NTT, or a schoolbook polynomial multiplication for the incomplete NTT. Similarly, the inverse of a polynomial $a \in \mathcal{R}_{q,n}$ can be computed via

$$a^{-1} = \mathbf{INTT}_q^n\left(\mathbf{BaseInv}_q^n\left(\mathbf{NTT}_q^n(a)\right)\right),$$

where $\mathbf{BaseInv}_q^n$ computes pointwise inverses in $\mathbb{Z}_q$ for the complete NTT, or applies a schoolbook polynomial inversion for the incomplete NTT.

3 LARK Pseudorandom Number Generator

In this section, we present the detailed algorithmic description, probability analysis, and parameter selection of the lattice-based PRG scheme LARK.

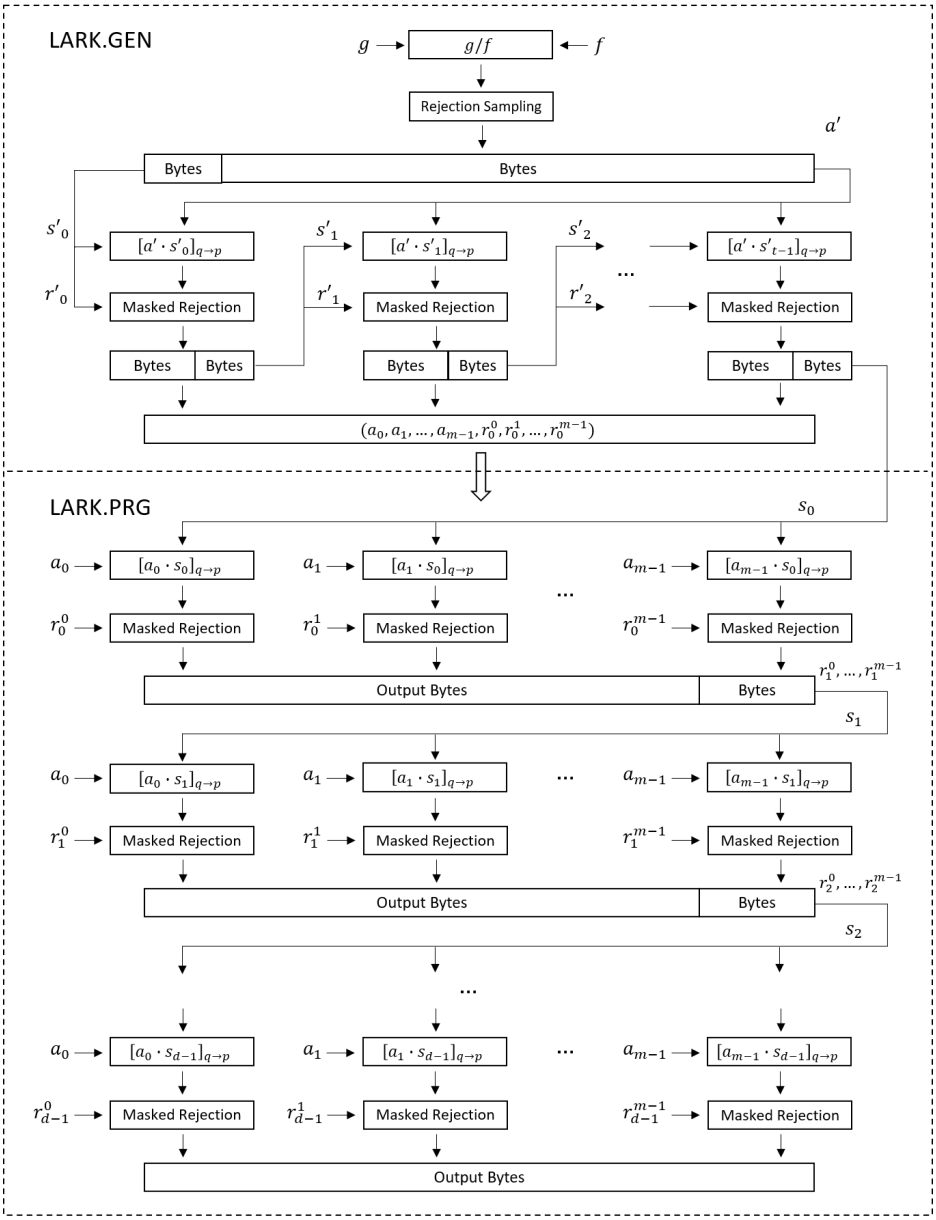

Figure 1: The Structure of LARK

Our structural design is built upon the framework introduced in LEAP [ZLL+25]. More precisely, LARK inherits from LEAP the idea of treating the public parameter as part of the key in the PRG setting, the two-phase GEN/PRG framework for key-size compression, and the reuse framework that simultaneously reuses a and s. The main differences are that LARK replaces the RLWR-based component with the RLAM-based formulation, uses the more compact parameter regime to replace the vectorized-NTRU assumption in LEAP with the standard NTRU assumption, and introduces a masked-rejection technique tailored to modulus reduction.

The scheme consists of two algorithms: the key generation algorithm LARK.GEN and the pseudorandom number generator algorithm LARK.PRG. In LARK.PRG, a fixed polynomial vector $\mathbf{a} \in \mathcal{R}_{q,n}^m$ is reused across all iterations, while the polynomial s_j and mask vector $\mathbf{r}_j$ are updated at each iteration. In each iteration, the values $([a_0 \cdot s_j]_{q\to p}, \ldots, [a_{m-1} \cdot s_j]_{q\to p})$ are computed, where part of the output is used as pseudorandom bytes and the remaining part is used to sample the next $(s_{j+1}, \mathbf{r}_{j+1})$. By reusing both $\mathbf{a}$ and s under the RLAM formulation, the structure significantly reduces computational overhead, since the NTT representation of s needs to be computed only once per iteration instead of once per multiplication. To mitigate the large key size resulting from treating public parameters as key material, LARK.GEN follows the GEN/PRG split of LEAP and compresses the key by using a single polynomial a' and iteratively updating $(s', \mathbf{r}')$ to derive $(a_0, a_1, \ldots, a_{m-1}, s_0, \mathbf{r}_0^0, \mathbf{r}_0^1, \ldots, \mathbf{r}_0^{m-1})$ for LARK.PRG. In contrast to LEAP, the compactness enabled by the RLAM formulation allows LARK.GEN to rely on the standard NTRU assumption rather than the vectorized-NTRU assumption, further reducing the key size. The overall structure of LARK is illustrated in Figure 1.

3.1 Algorithm Description

Public Parameters Before presenting the scheme in detail, we first specify the public parameters.

- N, Q: degree and modulus defining the ring $\mathcal{R}_{Q,N} = \mathbb{Z}_Q[x]/(x^N + 1)$, used in NTRU
- n, q: degree and modulus defining the ring $\mathcal{R}_{q,n} = \mathbb{Z}_q[x]/(x^n + 1)$, used in LAM
- v_f, v_g, v_s: bit-length parameters defining uniform distribution over $\mathcal{R}_{2^{v_f},N}$, $\mathcal{R}_{2^{v_g},N}$, $\mathcal{R}_{2^{v_s},n}$, respectively
- Q': rejection bound used in coefficient-wise rejection sampling over $\mathbb{Z}_Q$
- m: dimension of the polynomial vector $\mathbf{a} \in \mathcal{R}_{q,n}^m$
- p: modulus used in the modulus reduction function
- v_r: the bit length of the mask vector elements, thereby determining a rejection probability of 2^{-v_r}
- k: dimension of the mask vector for each polynomial
- l: length of the output pseudorandom byte array

In practical PRGs, pseudorandom values are typically packed into byte arrays. Under the NTRU assumption, however, the resulting elements are uniformly distributed over $\mathcal{R}_{Q,N}$, with coefficients uniformly distributed over $\mathbb{Z}_Q$. To enable efficient polynomial operations via the NTT, the modulus Q is chosen to be prime. Consequently, converting uniformly distributed elements in $\mathcal{R}_{Q,N}$ into uniformly distributed byte arrays requires a secure and efficient transformation from $\mathbb{Z}_Q$ to $\mathbb{Z}_{256}$, which is nontrivial. After exploring several alternatives, we found rejection sampling to be the most efficient and practical

solution. Rejection sampling is employed to transform uniformly distributed samples over a large domain into uniformly distributed samples over a smaller target domain while preserving exact uniformity. Given a uniformly random element $x \in \mathbb{Z}_Q$, the standard rejection procedure accepts x if $x < Q'$ and rejects it otherwise. Conditioned on acceptance, the resulting distribution is uniform over $\mathbb{Z}_{Q'}$. Both the lattice-based PRG schemes SPRING-RS [BDFK17] and LEAP [ZLL+25] employ this technique. The **RejectionSampling** algorithm (Algorithm 1) performs coefficient-wise rejection sampling on a polynomial $a \in \mathcal{R}_{Q,N}$. Given a bound $Q' < Q$, it outputs a vector $\mathbf{c} \in \mathbb{Z}_{Q'}^t$ consisting of all coefficients a_i such that $a_i < Q'$, where t denotes the number of accepted coefficients. Coefficients not satisfying the bound are discarded. Assuming a is uniformly distributed over $\mathcal{R}_{Q,N}$, the output is uniformly distributed over $\mathbb{Z}_{Q'}^t$.

Algorithm 1 RejectionSampling

Input: Integers Q', N; polynomial $a \in \mathcal{R}_{Q,N}$
Output: Length t and vector $\mathbf{c} \in \mathbb{Z}_{Q'}^t$

$t \leftarrow 0$
for $i = 0$ to $N - 1$ **do**
  if $a_i < Q'$ **then**
    $\mathbf{c}_t \leftarrow a_i$
    $t \leftarrow t + 1$
  end if
end for
return $(t, \mathbf{c})$

Algorithm 2 MaskedRejection

Input: Integers n, v; polynomial $a \in \mathcal{R}_{p,n}$; mask vector $\mathbf{r} \in \mathbb{Z}_{2^v}^k$
Output: Length t and vector $\mathbf{c} \in \mathbb{Z}_p^t$

$t \leftarrow 0$, $j \leftarrow 0$
for $i = 0$ to $n - 1$ **do**
  if $a_i = 0$ **then**
    if $\mathbf{r}_j \neq 0$ **then**
      $\mathbf{c}_t \leftarrow a_i$
      $t \leftarrow t + 1$
    end if ▷ Reject $a_i = 0$ with probability 2^{-v}
    $j \leftarrow j + 1$
  else
    $\mathbf{c}_t \leftarrow a_i$
    $t \leftarrow t + 1$
  end if
end for
return $(t, \mathbf{c})$

Unlike the NTRU component, the LAM implies that when q and p are coprime, the resulting elements are not uniformly distributed over $\mathcal{R}_{p,n}$. To address this issue, we introduce a mask vector that conditionally rejects certain coefficients according to the relationship between q and p, thereby restoring uniformity over $\mathbb{Z}_p^t$. For efficiency, we choose parameters satisfying $q = zp + 1$ and $1/(z + 1) = 2^{-v}$. This choice is primarily efficiency-driven rather than a security requirement. Under the condition $q = zp + 1$, the bias introduced by modulus reduction is concentrated only at the coefficient value 0: the coefficient equals 0 with probability $(z + 1)/q$, while each nonzero value occurs with

probability z/q. Hence, exact uniformity over $\mathbb{Z}_p$ can be restored by rejecting the value 0 with probability $1/(z+1)$. If the condition $q = zp + 1$ is relaxed, the bias generally spreads across multiple coefficient values. In principle, such non-uniformity can still be corrected by a suitable rejection procedure, but the correction becomes more complicated and less efficient. For the high-throughput PRG setting considered in this paper, the choice $q = zp + 1$ therefore provides the most practical uniformization mechanism. Moreover, when $1/(z+1) = 2^{-v}$, the mask vector can be generated efficiently by sampling from the power-of-two distribution $U(\{0, 1, \ldots, 2^v - 1\}^k)$, which can be implemented using inexpensive operations. Algorithm 2 implements this masked rejection procedure for a polynomial $a \in \mathcal{R}_{p,n}$. For each coefficient a_i, the algorithm always accepts a_i whenever $a_i \neq 0$. When $a_i = 0$, acceptance is determined by a corresponding mask value $\mathbf{r}_j \in \mathbb{Z}_{2^v}$: the coefficient is accepted if and only if $\mathbf{r}_j \neq 0$. Since the mask vector is uniformly distributed over $\mathbb{Z}_{2^v}^k$, a coefficient equal to 0 is rejected with probability exactly 2^{-v}. The algorithm outputs the accepted coefficients together with their total number, which may be smaller than n. With an appropriate choice of parameters, the probability that a mask vector of length k is insufficient for a polynomial of dimension n is exponentially small (see Section 3.2.2).

We next define two sampling subroutines: **SamplePoly2$^{\mathbf{v}}$** (Algorithm 3) and **SamplePolyq** (Algorithm 4). Both algorithms first interpret the input byte array $\mathbf{b}$ as a bit stream $\mathbf{c}$. The **SamplePoly2$^{\mathbf{v}}$** algorithm parses $\mathbf{c}$ into consecutive v-bit blocks, each interpreted as a coefficient in $\mathbb{Z}_{2^v}$. When $\mathbf{b}$ is uniformly random, the resulting polynomial is uniformly distributed over $\mathcal{R}_{2^v,n}$. The **SamplePolyq** algorithm sets $k = \lceil \log_2 q \rceil$ and parses $\mathbf{c}$ into consecutive k-bit blocks. Each block is interpreted as an integer $d \in [0, 2^k)$ and accepted if $d < q$; otherwise, it is discarded. When $\mathbf{b}$ is uniformly random, this rejection procedure yields a polynomial uniformly distributed over $\mathcal{R}_{q,n}$. In practice, the bit stream need not be explicitly constructed, since the required bit blocks can be extracted directly from the byte array using inexpensive byte accesses, bit shifts, and masks.

Algorithm 3 SamplePoly2$^{\mathbf{v}}$

Input: Integers v, n with $v \leq 8$ and $8 \mid vn$; byte array $\mathbf{b} \in \mathbf{B}^{vn/8}$
Output: Polynomial $a \in \mathcal{R}_{2^v,n}$
$\mathbf{c}_{8i+j} \leftarrow \left\lfloor \mathbf{b}_i / 2^j \right\rfloor \bmod 2$ for all $0 \leq i < vn/8$ and $0 \leq j < 8$ ▷ $\mathbf{c}$ is the bit stream of $\mathbf{b}$
for $i = 0$ to $n - 1$ **do**
  $a_i \leftarrow \sum_{j=0}^{v-1} c_{iv+j} 2^j$
end for
return a

Algorithm 4 SamplePolyq

Input: Integers q, n; byte array $\mathbf{b} \in \mathbf{B}^L$ with sufficient length
Output: Polynomial $a \in \mathcal{R}_{q,n}$
$k \leftarrow \lceil \log_2 q \rceil$
$\mathbf{c}_{8i+j} \leftarrow \left\lfloor \mathbf{b}_i / 2^j \right\rfloor \bmod 2$ for all $0 \leq i < L$ and $0 \leq j < 8$ ▷ $\mathbf{c}$ is the bit stream of $\mathbf{b}$
$u \leftarrow 0$
for $i = 0$ to $n - 1$ **do**
  repeat
    $d \leftarrow \sum_{j=0}^{k-1} c_{u+j} 2^j$
    $u \leftarrow u + k$
  until $d < q$
  $a_i \leftarrow d$
end for
return a

Algorithm 5 LARK.GEN

Input: Public parameters; polynomials $f \in \mathcal{R}_{2^{v_f},N}$, $g \in \mathcal{R}_{2^{v_g},N}$ with $f \in \mathcal{R}^*_{Q,N}$
Output: Polynomial vector $\hat{\mathbf{a}} \in \mathcal{R}^m_{q,n}$ (NTT domain); polynomial $s \in \mathcal{R}_{2^{v_s},n}$; mask vector $\mathbf{r} \in \mathbb{Z}^{km}_{2^{v_r}}$

1: $d_r \leftarrow v_r k/8$, $d_r^m \leftarrow v_r km/8$, $d_s \leftarrow v_s n/8$
2: $\hat{f} \leftarrow \mathbf{NTT}^N_Q(f)$
3: $\hat{f}^{-1} \leftarrow \mathbf{BaseInv}^N_Q(\hat{f})$
4: $\hat{g} \leftarrow \mathbf{NTT}^N_Q(g)$
5: $\hat{h} \leftarrow \mathbf{BaseMul}^N_Q(\hat{g}, \hat{f}^{-1})$
6: $(t, \mathbf{z}) \leftarrow \mathbf{RejectionSampling}(Q', N, \hat{h})$
7: $\mathbf{r}' \leftarrow \mathbf{SamplePoly2^v}(v_r, k, \mathbf{z}_{t-d_r:t-1})$
8: $t \leftarrow t - d_r$
9: $s' \leftarrow \mathbf{SamplePoly2^v}(v_s, n, \mathbf{z}_{t-d_s:t-1})$
10: $t \leftarrow t - d_s$
11: $\hat{a}' \leftarrow \mathbf{SamplePolyq}(q, n, \mathbf{z}_{0:t-1})$
12: $u \leftarrow 0$
13: **repeat**
14: $\quad \hat{s}' \leftarrow \mathbf{NTT}^n_q(s')$
15: $\quad \hat{h} \leftarrow \mathbf{BaseMul}^n_q(\hat{a}', \hat{s}')$
16: $\quad h \leftarrow \mathbf{INTT}^n_q(\hat{h})$
17: $\quad$ **for** $j = 0$ to $n-1$ **do**
18: $\quad\quad h_j = [h_j]_{q \to p}$
19: $\quad$ **end for**
20: $\quad (t, \mathbf{c}) \leftarrow \mathbf{MaskedRejection}(n, v_r, h, \mathbf{r}')$
21: $\quad \mathbf{b}_{0:u+t-1} \leftarrow \mathbf{b}_{0:u-1} || \mathbf{c}$
22: $\quad u \leftarrow u + t$
23: $\quad \mathbf{r}' \leftarrow \mathbf{SamplePoly2^v}(v_r, k, \mathbf{b}_{u-d_r:u-1})$
24: $\quad u \leftarrow u - d_r$
25: $\quad s' \leftarrow \mathbf{SamplePoly2^v}(v_s, n, \mathbf{b}_{u-d_s:u-1})$
26: $\quad u \leftarrow u - d_s$
27: **until** $\mathbf{b}$ contains a sufficient number of bytes to sample the polynomial vector $\hat{\mathbf{a}}$ and mask vector $\mathbf{r}$
28: $\mathbf{r} \leftarrow \mathbf{SamplePoly2^v}(v_r, km, \mathbf{b}_{u-d_r^m:u-1})$
29: $u \leftarrow u - d_r^m$
30: $\hat{\mathbf{a}} \leftarrow \mathbf{SamplePolyq}(q, mn, \mathbf{b}_{0:u-1})$
31: $s \leftarrow s'$
32: **return** $(\hat{\mathbf{a}}, s, \mathbf{r})$.

Algorithm 5 specifies a deterministic procedure for generating the key $(\mathbf{a}, s, \mathbf{r}) \in \mathcal{R}^m_{q,n} \times \mathcal{R}_{2^{v_s},n} \times \mathbb{Z}^{km}_{2^{v_r}}$ (each polynomial consumes a mask vector of length k, hence m polynomials require km mask entries) used by the LARK.PRG algorithm. The algorithm proceeds in two main phases. In the first phase, a byte array is generated based on the NTRU assumption (Steps 2–6). To improve efficiency, this byte generation is carried out directly in the NTT domain $\hat{h}$ (Step 6). Since the NTT is a bijective linear transformation over $\mathcal{R}_Q$, it preserves uniformity. Hence, if h is uniformly distributed over $\mathcal{R}_Q$, then its NTT representation $\hat{h}$ is also uniformly distributed over $\mathcal{R}_Q$. Since the inverse NTT is a deterministic and publicly computable mapping, applying it does not change the distribution. Any adversary can efficiently recompute the NTT of h to recover its NTT representation. Therefore, the inverse NTT step can be safely omitted without affecting security, resulting in a more efficient implementation. The resulting uniformly distributed bytes are then used to sample

the tuple $(a', s', \mathbf{r}') \in (\mathcal{R}_{q,n} \times \mathcal{R}_{2^{v_s},n} \times \mathbb{Z}_{2^{v_r}}^k)$. By appropriately choosing parameters, we ensure that the probability that the bytes generated from the NTRU-based phase are insufficient to support the required sampling procedures is exponentially small (see Section 3.2.2). In the second phase, additional bytes are generated under the RLAM assumption using the sampled tuple $(a', s', \mathbf{r}')$. Because the RLAM problem requires q and p to be coprime, the resulting output does not initially follow a uniform byte distribution. Based on the chosen parameters, the output is therefore conditionally rejected through a mask vector, which ultimately yields a uniformly random byte array. These bytes are divided into two parts: one part is used to resample updated values of s' and $\mathbf{r}'$, while the remaining part is appended to the output byte array. This process is iterated until a sufficient number of bytes has been accumulated to sample the polynomial vector $\mathbf{a}$ and mask vector $\mathbf{r}$ (Steps 13–27). The final unused value s' is output as the key s. Following the design of Kyber [BDK+18], both a' (Step 11) and $\mathbf{a}$ (step 30) are sampled directly in the NTT domain, avoiding unnecessary forward NTT computations. As a result, LARK.GEN outputs $\hat{\mathbf{a}}$ already in the NTT domain, eliminating the need for additional NTT transformations in LARK.PRG and further reducing computational overhead.

Algorithm 6 LARK.PRG

Input: Public parameters; polynomial vector $\hat{\mathbf{a}} \in \mathcal{R}_{q,n}^m$ (NTT domain); polynomial $s \in \mathcal{R}_{2^{v_s},n}$; mask vector $\mathbf{r} \in \mathbb{Z}_{2^{v_r}}^{km}$
Output: Byte array $\mathbf{b} \in \mathbf{B}^l$

1: $u \leftarrow 0$
2: $d_s \leftarrow v_s n/8$, $d_r^m \leftarrow v_r km/8$
3: **while** $u < l$ **do**
4: $\quad \hat{s} = \mathbf{NTT}_q^n(s)$
5: $\quad$ **for** $i = 0$ to $m-1$ **do**
6: $\quad\quad \hat{h} \leftarrow \mathbf{BaseMul}_q^n(\hat{\mathbf{a}}_i, \hat{s})$
7: $\quad\quad h \leftarrow \mathbf{INTT}_q^n(\hat{h})$
8: $\quad\quad$ **for** $j = 0$ to $n-1$ **do**
9: $\quad\quad\quad h_j = [h_j]_{q \to p}$
10: $\quad\quad$ **end for**
11: $\quad\quad (t, \mathbf{c}) \leftarrow \mathbf{MaskedRejection}(n, v_r, h, \mathbf{r}_{ik:(i+1)k-1})$
12: $\quad\quad \mathbf{b}_{0:u+t-1} \leftarrow \mathbf{b}_{0:u-1} || \mathbf{c}$
13: $\quad\quad u \leftarrow u + t$
14: $\quad$ **end for**
15: $\quad \mathbf{r} \leftarrow \mathbf{SamplePoly2^v}(v_r, km, \mathbf{b}_{u-d_r^m:u-1})$
16: $\quad u \leftarrow u - d_r^m$
17: $\quad s \leftarrow \mathbf{SamplePoly2^v}(v_s, n, \mathbf{b}_{u-d_s:u-1})$
18: $\quad u \leftarrow u - d_s$
19: **end while**
20: **return** $\mathbf{b}_{0:l-1}$.

Algorithm 6 defines a deterministic PRG based on the RLAM assumption. Its structure closely follows the second phase of LARK.GEN. Given a polynomial vector $\hat{\mathbf{a}}$ in the NTT domain and a polynomial s, the algorithm iteratively generates an output byte array. In each iteration, the polynomial s is first transformed into the NTT domain and then multiplied point-wise with each of the m independent polynomials $\hat{\mathbf{a}}_i$. The resulting products are transformed back to the coefficient domain via an inverse NTT. As in LARK.GEN, the generated bytes are divided into two parts: one part is used to resample s and $\mathbf{r}$, and the remaining part is appended to the output array. This procedure is repeated until l bytes have been generated, at which point the byte array is returned as the output of PRG.

3.2 Probability Analysis of Exceptional and Failure Events

In this section, we analyze the probabilities of the exceptional and failure events arising in LARK. We first consider the condition on the secret polynomial required in the subsequent security analysis of RLAM. For a modulus q, the secret polynomial s is required to satisfy $s \in \mathcal{R}_{q,n}^{\dagger}$, or equivalently, $\gcd(q, s_0, s_1, \ldots, s_{n-1}) = 1$. We therefore analyze the probability of sampling such a secret. We next analyze the failure probabilities associated with the rejection-based procedures in LARK. Rejection sampling and masked rejection may leave an insufficient number of pseudorandom bytes for subsequent sampling, thereby causing the corresponding procedure to terminate prematurely.

3.2.1 Exceptional Probability for the Secret Polynomial

We first consider the condition on the secret polynomials required in the subsequent security analysis. Specifically, each secret polynomial is required to satisfy $s \in \mathcal{R}_{q,n}^{\dagger}$, i.e., $\gcd(q, s_0, s_1, \ldots, s_{n-1}) = 1$. In our construction, the modulus q is chosen as a prime supporting efficient NTT computation. Hence, $\gcd(q, s_0, s_1, \ldots, s_{n-1}) \neq 1$ if and only if all coefficients of s are zero modulo q. Therefore, the exceptional probability for a single sampled secret polynomial is

$$\delta_s := \Pr\left[s \notin \mathcal{R}_{q,n}^{\dagger}\right] = \Pr[s_0 = s_1 = \cdots = s_{n-1} = 0].$$

During the execution of LARK, a total of $t + d$ secret polynomials are sampled, where t denotes the number of iterations in the RLAM-based phase of LARK.GEN, and d denotes the number of iterations in LARK.PRG. The overall exceptional probability that at least one of these secret polynomials does not belong to $\mathcal{R}_{q,n}^{\dagger}$ is

$$\delta_0 := 1 - (1 - \delta_s)^{t+d}.$$

3.2.2 Failure Probability of Masked Rejection

We now analyze the probability that the masked rejection procedure fails due to an insufficient mask vector. Specifically, failure occurs when the number of zero coefficients in a polynomial strictly exceeds the mask length k. Since each coefficient equals zero with a fixed probability determined by the modulus relation $q = zp + 1$, the number of zero coefficients follows a binomial distribution. The following lemma provides an exact expression for this failure probability.

Lemma 1. *Let n and q denote the degree and modulus of the* RLAM *ring, respectively, and let $p < q$ be coprime integers satisfying $q = zp + 1$. After modulus reduction, each coefficient satisfies*

$$\Pr[X = 0] = \frac{z+1}{q} \quad \textit{and} \quad \Pr[X \neq 0] = 1 - \frac{z+1}{q}.$$

Let k denote the length of the mask vector. The masked rejection procedure fails if the number of zero coefficients exceeds k. Then the failure probability is

$$\delta_{\text{mask}} \leq \sum_{i=k+1}^{n} \binom{n}{i} \left(\frac{z+1}{q}\right)^{i} \left(1 - \frac{z+1}{q}\right)^{n-i}.$$

Proof. In the RLAM problem, each coefficient is obtained by modulus reduction from $\mathbb{Z}_q$ to $\mathbb{Z}_p$. Since $q = zp + 1$, the value 0 has exactly $z + 1$ preimages in q, while each nonzero value in $\mathbb{Z}_p$ has exactly z preimages. Consequently, $\Pr[X = 0] = \frac{z+1}{q}$. Let Y denote the number of zero coefficients in the resulting polynomial. As the coefficients

are independent, Y follows a binomial distribution with parameters $\big(n, (z+1)/q\big)$. The masked rejection procedure consumes one mask entry for each zero coefficient. If the number of zero coefficients exceeds the mask length k, the mask vector is insufficient and the algorithm fails. Therefore,

$$\delta_{\text{mask}} = \Pr[Y > k] \leq \sum_{i=k+1}^{n} \binom{n}{i} \left(\frac{z+1}{q}\right)^{i} \left(1 - \frac{z+1}{q}\right)^{n-i}.$$

□

3.2.3 Failure Probability of LARK

In the LARK scheme, the use of rejection sampling and masked rejection introduces a nonzero probability that the sampling procedures may fail due to an insufficient number of accepted coefficients. Such failures occur when the available pseudorandom bytes are exhausted before all required polynomials or mask vectors can be successfully generated. In particular, we consider three potential failure events:

1. In the NTRU-based phase of LARK.GEN, the bytes generated under the NTRU assumption are insufficient to sample $(a', s', \mathbf{r}')$ due to excessive rejections (Algorithm 5, Steps 7–11).
2. In the RLAM-based phase of LARK.GEN, the bytes generated in at least one iteration are insufficient to resample the updated polynomial s'_{i+1} and mask vector $\mathbf{r}'_{i+1}$ due to excessive rejections (Algorithm 5, Steps 23–26).
3. In LARK.PRG, the bytes generated in at least one iteration using $(a_0, a_1, \ldots, a_{m-1})$, s_i, and $\mathbf{r}_i$ are insufficient to resample the updated polynomial s_{i+1} and mask vector $\mathbf{r}_{i+1}$ due to excessive rejections (Algorithm 6, Steps 15–18).

We analyze the failure probabilities associated with each of these events in the following. To this end, we first define

$$\mathcal{P}(n_0, n_1, q_0, q_1) := \sum_{j=n_0-n_1}^{n_0} \binom{n_0}{j} \left(1 - \frac{q_1}{q_0}\right)^{j} \left(\frac{q_1}{q_0}\right)^{n_0-j}.$$

Lemma 2. *Let N and Q denote the degree and modulus of the* NTRU *ring, respectively, and let $Q' < Q$ be the rejection bound used in coefficient-wise rejection sampling. Let s_{byte} and r_{byte} denote the number of bytes required to sample the polynomial $s' \in \mathcal{R}_{2^{v_s},n}$ and the mask vector $\mathbf{r}' \in \mathbb{Z}^k_{2^{v_r}}$, respectively. The probability that the* NTRU*-based phase of* LARK.GEN *fails to generate the $(a', s', \mathbf{r}')$ is upper bounded by*

$$\delta_1 \;\leq\; \sum_{i=0}^{N} \left(\delta_i^{\text{NTRU}} \cdot \delta_i^{\text{fail}}\right),$$

where

$$\delta_i^{\text{NTRU}} \leq \binom{N}{i} \left(1 - \frac{Q'}{Q}\right)^{i} \left(\frac{Q'}{Q}\right)^{N-i}$$

denotes the probability that exactly i coefficients are rejected during the NTRU*-based rejection sampling, and*

$$\delta_i^{\text{fail}} := \begin{cases} 1, & \text{if } N - i < s_{\text{byte}} + r_{\text{byte}}, \\ \delta_i^{a'}, & \text{otherwise.} \end{cases}$$

In the latter case, define

$$A_i := \left\lfloor \frac{8\,(N - i - s_{\text{byte}} - r_{\text{byte}})}{\lceil \log_2 q \rceil} \right\rfloor,$$

which represents the number of candidate coefficients available for sampling $a' \in \mathcal{R}_{q,n}$ after reserving the bytes for s' and $\mathbf{r}'$. Then

$$\delta_i^{a'} := \begin{cases} 1, & \text{if } A_i < n, \\ \mathcal{P}\big(A_i,\, n-1,\, 2^{\lceil \log_2 q \rceil},\, q\big), & \text{otherwise.} \end{cases}$$

Proof. In the NTRU-based phase of LARK.GEN, each of the N coefficients is independently accepted with probability Q'/Q. Hence, the probability that exactly i coefficients are rejected is

$$\delta_i^{\text{NTRU}} \le \binom{N}{i} \left(1 - \frac{Q'}{Q}\right)^i \left(\frac{Q'}{Q}\right)^{N-i}.$$

After rejection sampling, $N - i$ coefficients remain. If $N - i < s_{\text{byte}} + r_{\text{byte}}$, the remaining bytes are insufficient to sample s' and $\mathbf{r}'$, and the algorithm fails deterministically. Otherwise, after reserving s_{byte} and r_{byte} bytes, the remaining data yield A_i candidate coefficients for generating a' via rejection sampling from $2^{\lceil \log_2 q \rceil}$ down to q. If $A_i < n$, the available candidates are insufficient to sample all coefficients of a', and the algorithm again fails deterministically. If $A_i \ge n$, failure occurs only when at least $A_i - n + 1$ candidates are rejected in the rejection sampling from $2^{\lceil \log_2 q \rceil}$ to q, which happens with probability $\delta_i^{a'}$. Summing over all possible values of i yields the stated upper bound on the failure probability δ_1. □

Lemma 3. *Let n and q denote the degree and modulus of the* RLAM *ring, respectively, and let $p < q$ be coprime integers satisfying $q = zp + 1$. Let s_{byte} and r_{byte} denote the number of bytes required to sample the polynomial $s' \in \mathcal{R}_{2^{v_s},n}$ and the mask vector $\mathbf{r}' \in \mathbb{Z}_{2^{v_r}}^k$, respectively. Let t denote the number of iterations in the* RLAM*-based phase of* LARK.GEN*. Then the probability that the* RLAM*-based phase of* LARK.GEN *fails to update s' and $\mathbf{r}'$ due to masked rejection is upper bounded by*

$$\delta_2 \;\le\; 1 - (1 - \delta_{\text{RLAM}})^t,$$

where δ_{RLAM} denotes the failure probability of a single iteration and satisfies

$$\delta_{\text{RLAM}} \le \mathcal{P}(n,\, s_{\text{byte}} + r_{\text{byte}} - 1,\, q,\, q-1) + \delta_{\text{mask}}.$$

Proof. In the RLAM problem, each coefficient is obtained by modulus reduction from $\mathbb{Z}_q$ to $\mathbb{Z}_p$. Since q and p are coprime and the parameters satisfy $q = zp + 1$, the induced distribution over $\mathbb{Z}_p$ is biased only at the value 0: specifically, $\Pr[X = 0] = \frac{z+1}{q}$, while each nonzero value occurs with probability $\frac{z}{q}$. To restore uniformity over $\mathbb{Z}_p$, a masked rejection procedure is applied. Nonzero coefficients are always accepted, whereas a zero coefficient is rejected with probability 2^{-v_r} using a uniformly sampled mask. Under the parameter choice $2^{-v_r} = \frac{1}{z+1}$, the overall rejection probability for each coefficient becomes

$$\Pr[\text{reject}] = \Pr[X = 0] \cdot \Pr[\text{reject} \mid X = 0] = \frac{z+1}{q} \cdot \frac{1}{z+1} = \frac{1}{q}.$$

Let i denote the number of rejected coefficients in a single iteration. Since coefficients are independent, the probability that exactly i coefficients are rejected is

$$\binom{n}{i} \left(\frac{1}{q}\right)^i \left(1 - \frac{1}{q}\right)^{n-i}.$$

After rejection, $n - i$ coefficients remain. If $n - i < s_{\text{byte}} + r_{\text{byte}}$, the remaining bytes are insufficient to sample the polynomial s' and the mask vector $\mathbf{r}'$, and the algorithm fails deterministically. Therefore, the failure probability of a single RLAM iteration is the probability that at least $n - (s_{\text{byte}} + r_{\text{byte}}) + 1$ coefficients are rejected, or that the number of zero coefficients exceeds the mask length. This probability is upper bounded by

$$\delta_{\text{RLAM}} \leq \mathcal{P}(n,\, s_{\text{byte}} + r_{\text{byte}} - 1,\, q,\, q-1) + \delta_{\text{mask}}.$$

Since the RLAM-based phase consists of t independent iterations, the probability that at least one iteration fails is upper bound by

$$\delta_2 \leq 1 - (1 - \delta^{\text{RLAM}})^t.$$

□

Lemma 4. *Let n and q denote the degree and modulus of the* RLAM *ring, respectively, and let $p < q$ be coprime integers satisfying $q = zp + 1$. Let m denote the dimension of the polynomial vector $(a_0, a_1, \ldots, a_{m-1})$. Let s_{byte} and r_{byte} denote the number of bytes required to sample the polynomial $s \in \mathcal{R}_{2^{v_s},n}$ and the mask vector $\mathbf{r} \in \mathbb{Z}_{2^{v_r}}^{km}$, respectively. Let d denote the number of iterations in the* LARK.PRG*. Then the probability that the* LARK.PRG *fails to update s and $\mathbf{r}$ due to masked rejection is upper bounded by*

$$\delta_3 \;\leq\; 1 - (1 - \delta_{\text{RLAM}}^m)^d\,,$$

where the failure probability of a single iteration satisfies

$$\delta_{\text{RLAM}}^m \leq \mathcal{P}(mn,\, s_{\text{byte}} + r_{\text{byte}} - 1,\, q,\, q-1) + (1 - (1 - \delta_{\text{mask}})^m).$$

Proof. As discussed in Lemma 3, each coefficient is rejected with probability $\Pr[\text{reject}] = \frac{1}{q}$. In LARK.PRG, a single iteration using $(a_0, a_1, \ldots, a_{m-1})$, s_i, and $\mathbf{r}_i$ produces mn coefficients in total. Let i denote the number of rejected coefficients in one iteration. The probability that exactly i coefficients are rejected is

$$\binom{mn}{i} \left(\frac{1}{q}\right)^i \left(1 - \frac{1}{q}\right)^{mn-i}.$$

After rejection, $mn - i$ coefficients remain. If $mn - i < s_{\text{byte}} + r_{\text{byte}}$, the remaining bytes are insufficient to update the polynomial s and the mask vector $\mathbf{r}$, and the algorithm fails deterministically. Hence, the probability that a single iteration fails due to insufficient accepted coefficients is $\mathcal{P}(mn,\, s_{\text{byte}} + r_{\text{byte}} - 1,\, q,\, q-1)$. In addition, failure may occur if, for at least one of the m polynomials, the number of zero coefficients exceeds the available mask entries. Assuming independence across the m polynomials, this event is upper bounded by $1 - (1 - \delta_{\text{mask}})^m$. By the union bound, the failure probability of a single iteration satisfies

$$\delta_{\text{RLAM}}^m \leq \mathcal{P}(mn,\, s_{\text{byte}} + r_{\text{byte}} - 1,\, q,\, q-1) + (1 - (1 - \delta_{\text{mask}})^m).$$

Finally, since the LARK.PRG consists of d independent iterations, the probability that at least one iteration fails is

$$\delta_3 \;\leq\; 1 - (1 - \delta_{\text{RLAM}}^m)^d\,.$$

□

3.3 Parameter Selection

Table 2 presents the recommended parameters for LARK at the 128-bit and 256-bit security levels. The exceptional probability δ_0 arises from the secret-sampling distribution. Since the secret polynomial is sampled uniformly from $\mathcal{R}_{4,128}$ and $\mathcal{R}_{16,128}$ for the 128-bit and 256-bit security levels, respectively, and q is prime, the event $s \notin \mathcal{R}^{\dagger}_{q,n}$ occurs only when s is the all-zero polynomial. Hence, $\delta_s = 2^{-256}$ and $\delta_s = 2^{-512}$ for the two security levels, respectively.

Table 2: Parameters Sets for LARK

Scheme	Key	N	Q	Q'	v_f	v_g	n	q	p	v_r	k	v_s
LARK-128	128	512	257	256	1	1	128	769	256	2	32	2
LARK-256	256	1024	257	256	1	1	128	769	256	2	52	4

For each target security level $\lambda \in \{128, 256\}$ and polynomial vector dimension m, we define the maximum supported iteration count $d_{\max}(\lambda, m)$ as the largest integer d satisfying $\delta_0 + \delta_1 + \delta_2 + \delta_3 < 2^{-\lambda}$, where the exceptional and failure probabilities are evaluated as described in Section 3.2. In each LARK.PRG iteration, mn coefficients are generated, each of which is accepted with probability $1 - 1/q$ under the ideal uniform model. After reserving $nv_s/8$ bytes for the secret polynomial and $mkv_r/8$ bytes for the mask vector, the expected number of output bytes per iteration is

$$\mu_\lambda(m) = mn\left(1 - \frac{1}{q}\right) - \frac{nv_s}{8} - \frac{mkv_r}{8}.$$

Accordingly, the corresponding expected output length is $\bar{\ell}_{\max}(\lambda, m) = \lfloor d_{\max}(\lambda, m)\mu_\lambda(m) \rfloor$. Table 3 reports $d_{\max}(\lambda, m)$ and $\bar{\ell}_{\max}(\lambda, m)$ for the selected parameter sets.

Table 3: Maximum Supported Iterations and Expected Output Lengths

m	LARK-128		LARK-256	
	$d_{\max}$	$\ell_{\max}$ (bytes)	$d_{\max}$	$\ell_{\max}$ (bytes)
1	39511	3470391	290866	14785751
8	4938	4575888	36358	31074033
16	2469	4654896	18179	32237489
32	1234	4692499	9089	32817412
48	823	4707568	6059	33009495
64	617	4712243	4544	33104617
128	308	4714461	2272	33250025
256	154	4719389	1136	33322729
512	77	4721853	568	33359081

The reported output lengths are expectations rather than deterministic guarantees, since the number of accepted coefficients varies across iterations. In our concrete security evaluation, the maximum supported iteration count $d_{\max}(\lambda, m)$ is determined by requiring the total probability of exceptional and failure events to remain below $2^{-\lambda}$ for the selected parameter sets. Accordingly, the estimated 128-bit and 256-bit concrete security levels of LARK apply to executions satisfying $d \le d_{\max}(\lambda, m)$. Consequently, beyond the maximum supported iteration count reported in Table 3, we do not claim that potential degradation in pseudorandomness remains negligible.

4 Security

In this section, we present the security analysis of LARK, covering theoretical hardness foundations, formal security reductions, and concrete security evaluation. We first formalize the Ring Learning with Alternating Moduli (RLAM) problem as the ring analogue of LAM and establish its hardness via a reduction from RLWE under suitable parameter conditions. We then establish the pseudorandomness of LARK under the decisional NTRU and RLAM assumptions, with a formal security bound that accounts for reduction losses and exceptional events. Finally, we evaluate the concrete security of LARK by estimating the costs of the best-known attacks against the lattice instances arising from its concrete construction and parameter sets.

It is important to distinguish theoretical hardness reductions from the concrete security evaluation of practical lattice-based constructions. It is not uncommon in practical lattice-based cryptography for theoretical hardness reductions to impose sufficient parameter conditions that are more restrictive than those adopted in efficient implementations. In such cases, the hardness of the concrete instances is assessed through cryptanalysis rather than established directly by those reductions. By contrast, a security reduction from breaking a cryptographic construction to solving its underlying hard problems establishes the mathematical connection between the security of the construction and the hardness of those problems. In practical lattice-based cryptography, concrete security levels are therefore typically estimated from the computational costs of the best-known attacks against the relevant lattice instances, while also considering attacks that exploit the specific structure of the construction. These attack-cost estimates are distinct from the distinguishing-advantage bounds obtained through formal security reductions. In particular, losses in reduction tightness must be accounted for when deriving a formal advantage bound, but do not, by themselves, imply corresponding improvements in concrete attack complexity. This distinction is also reflected in the NIST-standardized ML-KEM, whose practical security strengths are assessed through concrete cryptanalysis of the underlying Module-LWE instances rather than derived directly from its formal security reductions [Nat24b]. Following this established methodology, our concrete security analysis accounts for the algebraic structure of LARK's key-generation and output-generation mechanisms, including the information available to the adversary and the relations arising from polynomial reuse across iterations, and estimates the costs of known attacks against the resulting lattice instances.

4.1 Ring Learning with Alternating Moduli

We first define the Ring Learning With Alternating Moduli (RLAM) problem, which is the ring analogue of LAM.

Definition 6. Let $n \geq 1$ be the main security parameter and let $q > p \geq 2$ be coprime integers.

- Let $\mathcal{X}_s$ be a distribution over $\mathcal{R}_{q,n}$. For a secret $s \leftarrow \mathcal{X}_s$, the decisional RLAM problem, denoted by $\mathrm{DRLAM}_{m,n,q,p,\mathcal{X}_s}$, is to distinguish between the following two distributions given $m = \mathrm{poly}(n)$ independent samples:

$$(a_i, b_i = [a_i \cdot s]_{q\to p}) \quad \text{and} \quad (a_i, [u_i]_{q\to p}),$$

  for $i \in \{0, \ldots, m-1\}$, where $a_i \leftarrow U(\mathcal{R}_{q,n})$ and $u_i \leftarrow U(\mathcal{R}_{q,n})$.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

- The advantage of an adversary $\mathcal{A}$ in distinguishing the decisional RLAM problem $\mathrm{DRLAM}_{m,n,q,p,\mathcal{X}_s}$ is defined as

$$\mathrm{Adv}^{\mathrm{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{A}) = |\Pr[\mathcal{A}((a_0, b_0), \ldots, (a_{m-1}, b_{m-1})) = 1]$$

$$- \Pr\left[\mathcal{A}((a_0, [u_0]_{q\to p}), \ldots, (a_{m-1}, [u_{m-1}]_{q\to p})) = 1\right] |.$$

The RLAM assumption states that for appropriate parameters (m, n, q, p) and secret distribution $\mathcal{X}_{\mathbf{s}}$, no probabilistic polynomial-time adversary can solve $\mathrm{DRLAM}_{m,n,q,p,\mathcal{X}_{\mathbf{s}}}$ with non-negligible advantage.

Lemma 5. *Let $m, n \geq 1$ with $m = \mathrm{poly}(n)$, and let $q > p \geq 2$ be coprime integers. Let $\mathcal{X}_s$ be any efficiently samplable distribution supported on $\mathcal{R}^{\dagger}_{q,n}$, and let $\mathcal{X}_e$ be the RLWE error distribution over $\mathcal{R}_{q,n}$. Let $B > 0$ satisfy $\Pr_{e\leftarrow\mathcal{X}_e}[\|e\|_\infty > B] = \mathsf{negl}(n)$, and assume $q > 2mnpB \cdot T$, where T is super-polynomial in n. If there exists a PPT algorithm $\mathcal{A}$ that solves $\mathrm{DRLAM}_{m,n,q,p,\mathcal{X}_s}$ with non-negligible advantage, then there exists a PPT algorithm $\mathcal{B}$ that solves $\mathrm{DRLWE}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}$ with non-negligible advantage.*

Proof sketch. The reduction follows Chen et al. [CJL26, Theorem 24]. For each RLWE sample (a_i, b_i), it computes $a'_i = pa_i \bmod q$ and $b'_i = pb_i \bmod q$, and gives $(a'_i, [b'_i]_{q\to p})$ to the RLAM distinguisher. Since $\gcd(p, q) = 1$, multiplication by p preserves uniformity, and hence the uniform case is mapped exactly to an RLAM no-instance. For an RLWE sample, we have $b'_i = a'_i s + pe_i \bmod q$. The additional step in the ring setting is to control coefficientwise wraparound. For $s \in \mathcal{R}^{\dagger}_{q,n}$, each coefficient of $a'_i s$ is a linear form whose coefficient vector is a signed cyclic permutation of $(s_0, \ldots, s_{n-1})$; therefore it is uniform over $\mathbb{Z}_q$ for uniform $a'_i \leftarrow \mathcal{R}_{q,n}$. Hence, each coefficient lies in the boundary region with probability at most $2pB/q$, and a union bound over the mn coefficients gives wraparound probability at most $2mnpB/q$, in addition to the negligible error-tail probability. Under the stated modulus condition, the resulting loss is negligible, so any non-negligible distinguisher for decisional RLAM yields a non-negligible distinguisher for decisional RLWE. The complete proof is given in Appendix A.

We emphasize that Lemma 5 establishes a theoretical hardness connection under its stated parameter conditions, which are not met by the practical LARK parameters. The concrete security estimates for these parameters are therefore based on the cryptanalytic evaluation in Section 4.3 rather than on a direct application of this lemma.

4.2 Security Reduction of LARK

Building on RLAM, we establish the security reductions for LARK. In particular, the security of LARK.GEN is reduced to the decisional NTRU and RLAM problems, while the security of LARK.PRG is reduced to the decisional RLAM problem. Consequently, any efficient adversary that distinguishes the output of LARK from the uniform distribution can be transformed into a distinguisher for the corresponding underlying lattice problems.

Theorem 1. *Let δ_0 denote the probability of sampling a secret polynomial $s \notin \mathcal{R}^{\dagger}_{q,n}$, and let δ_1, δ_2, and δ_3 denote the probabilities that* LARK *fails to produce the required output due to excessive rejections. If there exists a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ against* LARK*, then there exist a PPT adversary $\mathcal{B}$ against the decisional* NTRU *problem, a PPT adversary $\mathcal{C}_0$ against the single-sample decisional* RLAM *problem, and a PPT adversary $\mathcal{C}_1$ against the m-sample decisional* RLAM *problem, such that*

$$\mathrm{Adv}_{\mathrm{LARK}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{DNTRU}}_{N,Q,\mathcal{X}_g,\mathcal{X}_f}(\mathcal{B}) + t^2 \cdot \mathrm{Adv}^{\mathrm{DRLAM}}_{1,n,q,p,\mathcal{X}_s}(\mathcal{C}_0) + d^2 \cdot \mathrm{Adv}^{\mathrm{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{C}_1) + \delta_0 + \delta_1 + \delta_2 + \delta_3,$$

where t denotes the number of iterations in the RLAM*-based phase of* LARK.GEN*, and d denotes the number of iterations in* LARK.PRG*. Moreover, $\mathcal{B}$, $\mathcal{C}_0$, and $\mathcal{C}_1$ run in probabilistic polynomial time, with*

$$\mathsf{Time}(\mathcal{B}) = \mathsf{Time}(\mathcal{A}) + \mathrm{poly}(N, \log Q),$$

$$\mathsf{Time}(\mathcal{C}_0) = \mathsf{Time}(\mathcal{A}) + \mathrm{poly}(n, \log q),$$

$$\mathsf{Time}(\mathcal{C}_1) = \mathsf{Time}(\mathcal{A}) + \mathrm{poly}(n, m, \log q).$$

Proof. To prove Theorem 1, we define a sequence of games Game_0 through Game_6. The security of LARK is established by showing that Game_0 and Game_6 are computationally indistinguishable from the perspective of the adversary $\mathcal{A}$. The security reduction requires the sampled secret polynomial to satisfy $s \in \mathcal{R}^{\dagger}_{q,n}$. The probability that this condition is not satisfied is denoted by δ_0, and excluding this exceptional event introduces an additive loss of at most δ_0 in the distinguishing advantage. In addition, due to excessive rejection, LARK may fail to produce the required output with probabilities δ_1, δ_2, and δ_3. In the security reduction, we assume that the scheme always produces the required output successfully. The resulting increase in the distinguishing advantage caused by this assumption is bounded by $\delta_1 + \delta_2 + \delta_3$. Therefore, the total additive loss due to these exceptional events is bounded by $\delta_0 + \delta_1 + \delta_2 + \delta_3$, whose analysis is provided in Section 3.2.3.

Let $\text{Adv}_{\text{Game}}(\mathcal{A})$ denote the advantage of adversary $\mathcal{A}$ in a security game, defined as follows:

$$\text{Adv}_{\text{Game}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}(\mathbf{b} \leftarrow \text{Game}) = 1\right] - \Pr\left[\mathcal{A}(\mathbf{b} \leftarrow \mathbf{B}^l) = 1\right]\right|$$

For the LARK scheme, the security game is defined as follows. The challenger begins by sampling a random bit $b \in \{0, 1\}$. If $b = 0$, the challenger generates the byte array using LARK and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0, 1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

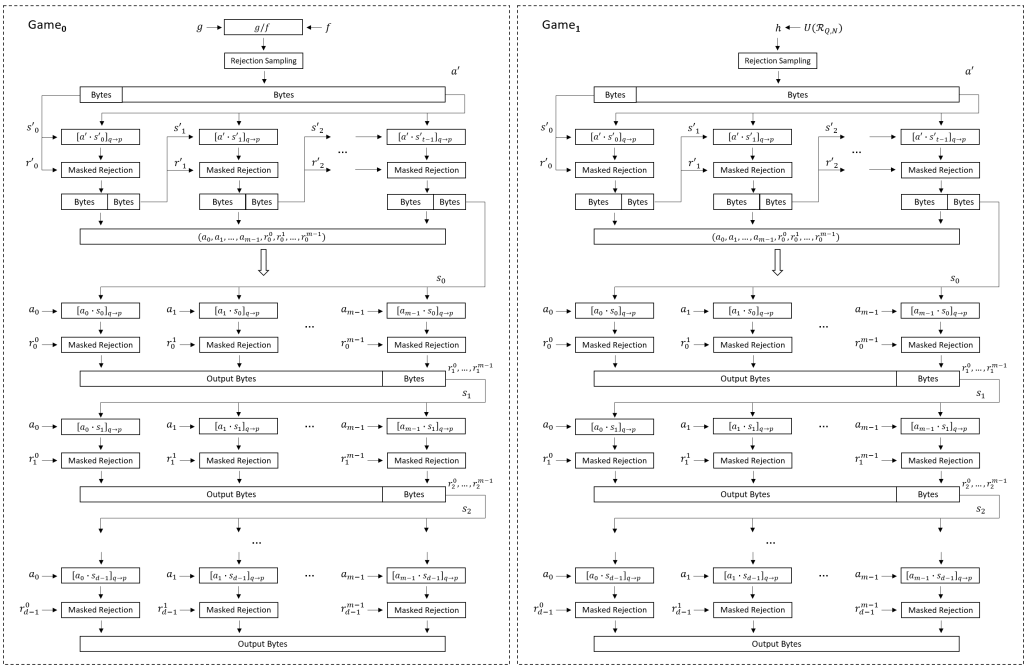

Figure 2: Game_0 and Game_1 for Theorem 1

- **Game_0**: The challenger begins by sampling a random bit $b \in \{0, 1\}$. If $b = 0$, the challenger generates the byte array according to Game_0 shown in Figure 2 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0, 1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

By definition, it holds that

$$\mathrm{Adv}_{\mathrm{Game}_0}(\mathcal{A}) = \mathrm{Adv}_{\mathrm{LARK}}(\mathcal{A}).$$

- **Game$_1$** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_1 shown in Figure 2 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_1 and Game_0 occurs in the NTRU-based phase. In Game_0, the bytes used to sample (a', s'_0, r'_0) are obtained by applying rejection sampling to the NTRU instance g/f, whereas in Game1 they are generated by directly sampling uniformly from $\mathcal{R}Q, N$ followed by the same rejection procedure. This change can only increase the distinguishing advantage of the adversary $\mathcal{A}$, and the increase is bounded by the advantage of solving the decisional NTRU problem. Formally, we have

$$|\mathrm{Adv}_{\mathrm{Game}_1}(\mathcal{A}) - \mathrm{Adv}_{\mathrm{Game}_0}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{DNTRU}}_{N,Q,\mathcal{X}_g,\mathcal{X}_f}(\mathcal{B}),$$

where $\mathcal{B}$ is an adversary against the decisional NTRU assumption.

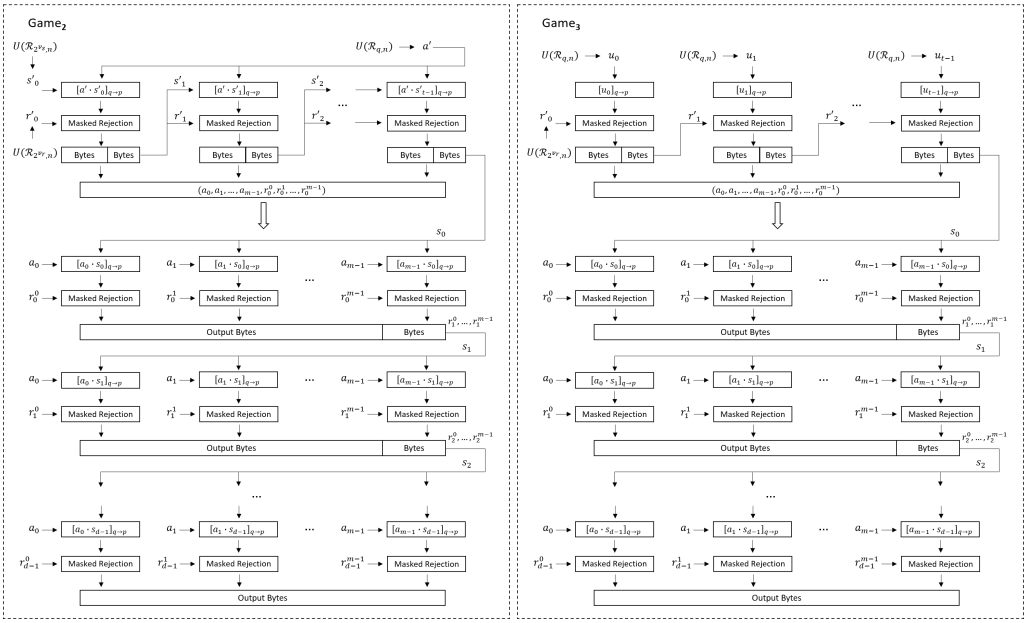

Figure 3: Game_2 and Game_3 for Theorem 1

- **Game$_2$** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_2 shown in Figure 3 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_2 and Game_1 lies in the procedure used to generate $(a', s'_0, \mathbf{r}'_0)$. In Game_1, elements are first sampled uniformly at random from $\mathcal{R}_{Q,N}$, after which rejection sampling is applied to obtain a uniformly random byte array. The tuple $(a', s'_0, \mathbf{r}'_0)$ is then derived from this byte array using Algorithms 4 and 3. In contrast,

Game_2 samples $(a', s'_0, \mathbf{r}'_0)$ directly from the target distribution $\mathcal{R}_{q,n} \times \mathcal{R}_{2^{v_s},n} \times \mathbb{Z}_{2^{v_r}}^k$. Since Algorithm 1 maps a uniformly random element of $\mathcal{R}_{Q,N}$ to a uniformly random byte array, the output bytes remain uniformly distributed whenever the input is uniform. Similarly, Algorithms 4 and 3 produce outputs following the desired target distributions when given uniformly random input bytes. Therefore, the joint distribution of $(a', s'_0, \mathbf{r}'_0)$ in Game_2 is identical to that in Game_1. Consequently, the view of the adversary remains unchanged, and we conclude that

$$\text{Adv}_{\text{Game}_2}(\mathcal{A}) = \text{Adv}_{\text{Game}_1}(\mathcal{A}).$$

- **Game$_3$** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_3 shown in Figure 3 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_3 and Game_2 is that, in Game_3, each term $[a' \cdot s'_j]_{q\to p}$ is replaced by $[u_j]_{q\to p}$, where $j \in \{0, \ldots, t-1\}$ and each u_j is sampled uniformly at random from $\mathcal{R}_{q,n}$. Thus, any change in the advantage of the adversary $\mathcal{A}$ arises solely from this modification. Let $\mathcal{A}'$ be an adversary that distinguishes the sequence $([a' \cdot s'_0]_{q\to p}, \ldots, [a' \cdot s'_{t-1}]_{q\to p})$ from $([u_0]_{q\to p}, \ldots, [u_{t-1}]_{q\to p})$. By definition, its distinguishing advantage equals $|\text{Adv}_{\text{Game}_3}(\mathcal{A}) - \text{Adv}_{\text{Game}_2}(\mathcal{A})|$. Applying a standard hybrid argument, there exist an index $j \in \{0, \ldots, t-1\}$ and an adversary $\mathcal{A}''$ distinguishes $\{[u_0]_{q\to p}, \ldots, [u_{j-1}]_{q\to p}, [a' \cdot s'_j]_{q\to p}, [a' \cdot s'_{j+1}]_{q\to p}, \ldots, [a' \cdot s'_{t-1}]_{q\to p}\}$ from $\{[u_0]_{q\to p}, \ldots, [u_{j-1}]_{q\to p}, [u_j]_{q\to p}, [a' \cdot s'_{j+1}]_{q\to p}, \ldots, [a' \cdot s'_{t-1}]_{q\to p}\}$ with advantage at least $1/t \cdot |\text{Adv}_{\text{Game}_3}(\mathcal{A}) - \text{Adv}_{\text{Game}_2}(\mathcal{A})|$. We now bound this advantage by reduction to the decisional RLAM problem. Construct an adversary $\mathcal{C}_0$ that receives an RLAM challenge (a, h), where h is either of the form $[a \cdot s]_{q\to p}$ for s, or $[u]_{q\to p}$ for $u \leftarrow U(\mathcal{R}_{q,n})$. The adversary $\mathcal{C}_0$ embeds h as the j-th component of the sequence, while generating the remaining components exactly as in the hybrid distributions above: the first j terms are computed as $[u]_{q\to p}$, and the remaining terms are computed as $[a' \cdot s']_{q\to p}$. The resulting sequence is then provided to $\mathcal{A}''$, whose output is used by $\mathcal{C}_0$ to decide the RLAM challenge. If $h = [a \cdot s]_{q\to p}$, the sequence corresponds to the hybrid distribution containing $[a' \cdot s'_j]_{q\to p}$; otherwise it corresponds to the one containing $[u_j]_{q\to p}$. Hence, any non-negligible advantage of $\mathcal{A}''$ directly translates into an advantage for $\mathcal{C}_0$ against RLAM. Since $\mathcal{C}_0$ does not know the correct index j, it guesses j uniformly at random, which succeeds with probability $1/t$. Therefore, $\text{Adv}(\mathcal{A}'') \leq t \cdot \text{Adv}^{\text{DRLAM}}_{1,n,q,p,\mathcal{X}_s}(\mathcal{C}_0)$, and combining the hybrid argument yields

$$|\text{Adv}_{\text{Game}_3}(\mathcal{A}) - \text{Adv}_{\text{Game}_2}(\mathcal{A})| \leq t^2 \cdot \text{Adv}^{\text{DRLAM}}_{1,n,q,p,\mathcal{X}_s}(\mathcal{C}_0).$$

- **Game$_4$** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_4 shown in Figure 4 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_4 and Game_3 lies in the procedure used to generate the tuple $(a_0, a_1, \ldots, a_{m-1}, s_0, \mathbf{r}_0^0, \mathbf{r}_0^1, \ldots, \mathbf{r}_0^{m-1})$. In Game_3, the elements $(u_0, u_1, \ldots, u_{t-1})$ are first sampled uniformly at random from $\mathcal{R}_{q,n}^t$, and $([u_0]_{q\to p}, [u_1]_{q\to p}, \ldots, [u_{t-1}]_{q\to p})$ are computed. Masked rejection is then applied to obtain a uniformly random byte array, from which the tuple $(a_0, a_1, \ldots, a_{m-1}, s_0, \mathbf{r}_0^0, \mathbf{r}_0^1, \ldots, \mathbf{r}_0^{m-1})$ is derived using Algorithms 4 and 3. In contrast, Game_4 samples $(a_0, a_1, \ldots, a_{m-1}, s_0, \mathbf{r}_0^0, \mathbf{r}_0^1, \ldots, \mathbf{r}_0^{m-1})$ directly from the target distribution $\mathcal{R}_{q,n}^m \times \mathcal{R}_{2^{v_s},n} \times \mathbb{Z}_{2^{v_r}}^{km}$. Since Algorithm 2 maps inputs following the specific distribution to a uniformly random byte array, the resulting bytes are uniformly distributed whenever the inputs satisfy the required distribution. Moreover, Algorithms 4 and 3 output

samples according to their intended target distributions when given uniformly random input bytes. Therefore, the joint distribution of $(a_0, a_1, \ldots, a_{m-1}, s_0, \mathbf{r}_0^0, \mathbf{r}_0^1, \ldots, \mathbf{r}_0^{m-1})$ in Game_4 is identical to that in Game_3. Consequently, the view of adversary remains unchanged, and we conclude that

$$\text{Adv}_{\text{Game}_4}(\mathcal{A}) = \text{Adv}_{\text{Game}_3}(\mathcal{A}).$$

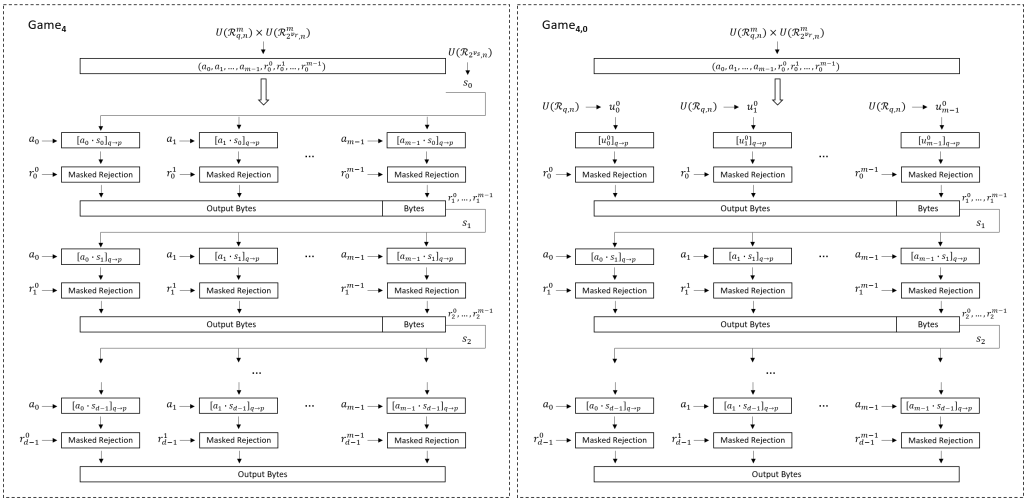

Figure 4: Game_4 and $\text{Game}_{4,0}$ for Theorem 1

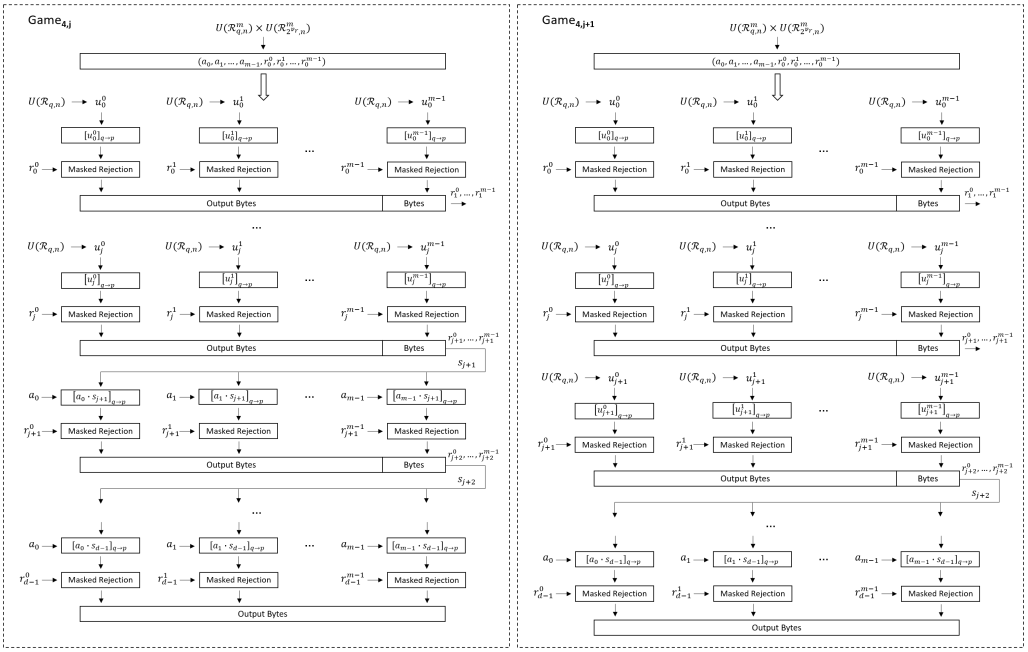

Figure 5: $\text{Game}_{4,j}$ and $\text{Game}_{4,j+1}$ for Theorem 1

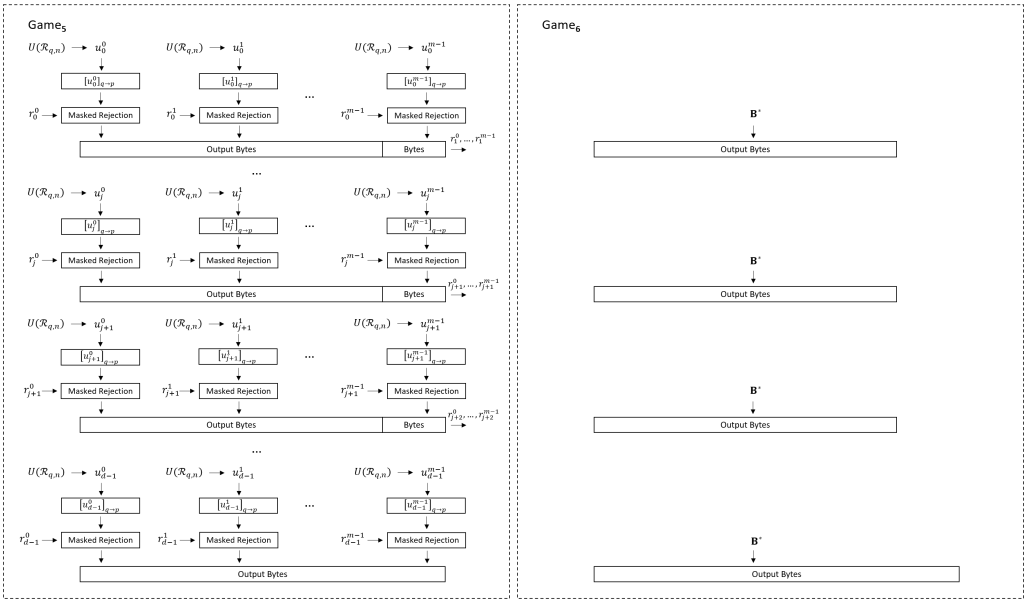

Figure 6: Game_5 and Game_6 for Theorem 1

- **Game_5** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_5 shown in Figure 6 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}^l$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_5 and Game_4 is that, in Game_5, each term $[a_i \cdot s_j]_{q\to p}$ is replaced by $[u^i_j]_{q\to p}$, where $i \in \{0,\ldots,m-1\}$ and $j \in \{0,\ldots,d-1\}$ and each u^i_j is sampled uniformly at random from $\mathcal{R}_{q,n}$. Thus, any change in the advantage of the adversary $\mathcal{A}$ arises solely from this modification. Let $\mathcal{A}'$ be an adversary that distinguishes between Game_5 and Game_4. By definition, its distinguishing advantage equals $|\text{Adv}_{\text{Game}_5}(\mathcal{A}) - \text{Adv}_{\text{Game}_4}(\mathcal{A})|$. We introduce a sequence of intermediate games $\text{Game}_{4,0},\ldots,\text{Game}_{4,d-1}$ (see Figure 4 and 5). In $\text{Game}_{4,j}$, the first $j+1$ columns are generated as in Game_5, while the remaining columns are generated as in Game_4. The only difference between $\text{Game}_{4,j}$ and $\text{Game}_{4,j+1}$ lies in the j-th column. By a standard hybrid argument, there exist an index $j \in \{0,\ldots,d-1\}$ and an adversary $\mathcal{A}''$ distinguishes $\text{Game}_{4,j}$ and $\text{Game}_{4,j+1}$ with advantage at least $1/d \cdot |\text{Adv}_{\text{Game}_5}(\mathcal{A}) - \text{Adv}_{\text{Game}_4}(\mathcal{A})|$. We now bound this advantage by reduction to the decisional RLAM problem. Construct an adversary $\mathcal{C}_1$ that receives an RLAM challenge $(a_0,h_0),\ldots,(a_{m-1},h_{m-1})$, where for each $i \in \{0,\ldots,m-1\}$, h_i is either of the form $[a \cdot s]_{q\to p}$ for s, or $[u]_{q\to p}$ for $u \leftarrow U(\mathcal{R}_{q,n})$. The adversary $\mathcal{C}_1$ embeds $h_0,\ldots,h_{m-1}$ as the j-th column of the matrix, while generating the remaining columns exactly as in the hybrid distributions above: the first j columns are computed as $[u]_{q\to p}$, and the remaining columns are computed as $[a \cdot s]_{q\to p}$. The resulting matrix is then provided to $\mathcal{A}''$, whose output is used by $\mathcal{C}_1$ to decide the RLAM challenge. If each $h_i = [a_i \cdot s]_{q\to p}$, the matrix is distributed as in $\text{Game}_{4,j}$; otherwise it is distributed as in $\text{Game}_{4,j+1}$. Hence, any non-negligible advantage of $\mathcal{A}''$ directly translates into an advantage for $\mathcal{C}_1$ against RLAM. Since $\mathcal{C}_1$ does not know the correct index j, it guesses j uniformly at random, which succeeds with probability $1/d$. Therefore, $\text{Adv}(\mathcal{A}'') \le d \cdot \text{Adv}^{\text{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{C}_1)$, and combining the hybrid argument yields

$$|\text{Adv}_{\text{Game}_5}(\mathcal{A}) - \text{Adv}_{\text{Game}_4}(\mathcal{A})| \le d^2 \cdot \text{Adv}^{\text{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{C}_1).$$

- **Game$_6$** : The challenger begins by sampling a random bit $b \in \{0,1\}$. If $b = 0$, the challenger generates the byte array according to Game_6 shown in Figure 6 and sends it to the adversary $\mathcal{A}$. If $b = 1$, the challenger samples a byte array uniformly at random from $\mathbf{B}'$ and sends it to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs a guess $b' \in \{0,1\}$ for the value of b. The challenger outputs 1 if $b' = b$ and 0 otherwise.

The only difference between Game_6 and Game_5 lies in the procedure used to generate the output bytes. In Game_5, the elements u_j^i, where $i \in \{0,\ldots,m-1\}$ and $j \in \{0,\ldots,d-1\}$, are first sampled uniformly at random from $\mathcal{R}_{q,n}$. Their reductions $[u_j^i]_{q\to p}$ are then computed, and masked rejection is applied to these values to obtain a uniformly random byte array, which serves as the output. In contrast, Game_6 samples output bytes directly and uniformly at random from $\mathbf{B}^*$. Since Algorithm 2 maps inputs following the specific distribution to a uniformly random byte array, the resulting bytes are uniformly distributed whenever the inputs satisfy the required distribution. It follows that the output distribution in Game_6 is identical to that in Game_5. Consequently, the view of adversary remains unchanged, and we conclude that $\text{Adv}_{\text{Game}_6}(\mathcal{A}) = \text{Adv}_{\text{Game}_5}(\mathcal{A})$. By definition, the output of Game_6 is perfectly uniform. Therefore,

$$\text{Adv}_{\text{Game}_6}(\mathcal{A}) = 0.$$

Based on the foregoing analysis, we derive the following upper bound on the advantage of any adversary $\mathcal{A}$ against LARK:

$$\mathbf{Adv}_{\text{LARK}}(\mathcal{A}) \le \text{Adv}^{\text{DNTRU}}_{N,Q,\mathcal{X}}(\mathcal{B}) + t^2 \cdot \text{Adv}^{\text{DRLAM}}_{1,n,q,p,\mathcal{X}_s}(\mathcal{C}_0) + d^2 \cdot \text{Adv}^{\text{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{C}_1) + \delta_0 + \delta_1 + \delta_2 + \delta_3.$$

□

Theorem 1 bounds the distinguishing advantage of LARK by the underlying NTRU and RLAM distinguishing advantages, with quadratic hybrid reduction losses associated with the GEN iteration count t and the PRG iteration count d, together with the exceptional and failure probabilities. For a given security parameter, the polynomial vector dimension m affects the underlying RLAM instance, the failure probabilities, and the expected output per PRG iteration. Consequently, the expected output length grows linearly with d, while the distinguishing advantage bound depends on t and d through the corresponding hybrid reduction losses and accumulated exceptional and failure probabilities. Thus, generating more output by increasing d also increases the iteration dependent terms in the security bound.

4.3 Concrete Security of LARK

We evaluate the concrete security of LARK by estimating the costs of the best-known attacks against the lattice instances arising from its specific construction and practical parameter sets. This attack-cost-based evaluation complements the theoretical reductions in Lemma 5 and Theorem 1, rather than relying on the sufficient parameter conditions of the former or directly instantiating the distinguishing-advantage bound of the latter. Importantly, our analysis is not limited to evaluating the underlying NTRU and RLAM problems in isolation. We also consider the specific algebraic structure of LARK, including the reuse of the polynomial vector $\mathbf{a}$ across iterations and the resulting relations among the outputs, as well as the information available to the adversary. These structural properties are incorporated into the corresponding lattice attack instances to assess their impact on the practical security of LARK. Following the standard methodology for concrete security evaluation in lattice-based cryptography, we use the state-of-the-art Lattice Estimator[4] from [APS15].

[4] https://github.com/malb/lattice-estimator, commit `3e48ef4`, `MATZOV` model

For NTRU, the estimator provides a dedicated module covering the standard attack landscape, including primal, hybrid, and overstretched lattice attacks and their optimized variants. Since LARK uses NTRU in its standard form, without introducing additional construction-specific structures or cross-instance dependencies, we directly apply the estimator to the corresponding NTRU instances and report the resulting bit-security estimates.

For RLAM, there is currently no specialized estimator. As is common in the concrete security analysis of structured lattice problems, such as RLWE and MLWE in ML-KEM [Nat24b] and ML-DSA [Nat24a], practical security is generally assessed by estimating the costs of the best-known attacks, often using corresponding unstructured LWE instances while considering known structure-specific attacks separately. For RLAM, we follow the recent analysis of LAM by Chen, Ji, and Li [CJL26], where LAM samples are transformed into LWE-like relations by treating the hidden integer quotients induced by modulus reduction as error terms. Specifically, we rewrite each RLAM relation as $b \equiv a \cdot s + pe \pmod{q}$, where e denotes the induced quotient. Since p is invertible modulo q, this relation can be transformed into an LWE-like instance with error e. The same algebraic transformation is used in [CJL26, Section 3.4, Theorem 29] and applies coefficientwise to ring products. Accordingly, we approximate the hardness of $\mathrm{DRLAM}_{n,q,p,\mathcal{X}_s}$ using the corresponding unstructured $\mathrm{LWE}_{n,q,\mathcal{X}_s,\mathcal{X}_e}$ instance and estimate its attack costs with the Lattice Estimator. While the algebraic transformation is exact, the concrete estimates are heuristic because the induced errors may depend on the matrix and secret, and the estimator approximates their distribution and correlations. Following LEAP [ZLL+25], LARK keeps the polynomial vector $\mathbf{a}$ secret and reuses it across iterations. To analyze attacks exploiting this reuse, we consider a stronger setting in which the adversary additionally observes all $b_{i,j}$ before masked rejection. For m polynomial samples over d iterations, the resulting relations can be written as

$$\begin{array}{ccc} [a_0 \cdot s_0]_{q\to p} = b_{0,0} & \cdots & [a_0 \cdot s_{d-1}]_{q\to p} = b_{0,d-1} \\ \vdots & \ddots & \vdots \\ [a_{m-1} \cdot s_0]_{q\to p} = b_{m-1,0} & \cdots & [a_{m-1} \cdot s_{d-1}]_{q\to p} = b_{m-1,d-1} \end{array}$$

Here each a_i is shared across columns, while each s_j is shared across rows. Although s_{j+1} is derived from the output of the preceding iteration, the following elimination holds for any pair of resulting states.

To analyze attacks exploiting the full multi-iteration transcript, we consider eliminating the shared a_i using multiple iteration columns. Let $\tau \in \{1, \ldots, d\}$ denote the number of selected iteration columns, where $\tau \geq 2$ is required for elimination. Using τ columns yields $(\tau - 1)mn$ coefficient relations involving τn secret coefficients. Following the primal attack analysis such as in [ADPS16], the resulting lattice has dimension $((\tau - 1)m + \tau)n$ and determinant $q^{(\tau-1)mn}$, and the estimated BKZ blocksize increases with τ for our parameters. Therefore, the two-column elimination ($\tau = 2$) achieves the lowest estimated cost within such eliminate-then-lattice-reduction model. For a fixed row i and two columns $j_1 \neq j_2$, we have

$$\begin{cases} a_i \cdot s_{j_1} + p \cdot e_{i,j_1} \equiv b_{i,j_1} \pmod{q}, \\ a_i \cdot s_{j_2} + p \cdot e_{i,j_2} \equiv b_{i,j_2} \pmod{q}, \end{cases}$$

and eliminating a_i yields

$$b_{i,j_1} s_{j_2} - b_{i,j_2} s_{j_1} \equiv p \cdot (e_{i,j_1} s_{j_2} - e_{i,j_2} s_{j_1}) \pmod{q}.$$

This yields the lattice

$$\mathcal{L}^{\mathrm{DRLAM}}_{j_1,j_2,q} = \begin{bmatrix} p^{-1}\cdot b_{0,j_1} & \cdots & p^{-1}\cdot b_{m-1,j_1} & 1 & 0 \\ p^{-1}\cdot b_{0,j_2} & \cdots & p^{-1}\cdot b_{m-1,j_2} & 0 & 1 \\ q & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & q & 0 & 0 \end{bmatrix},$$

where p^{-1} denotes the inverse of p in $\mathbb{Z}_q$. The vector

$$\left(e_{0,j_1}s_{j_2} - e_{0,j_2}s_{j_1},\ \ldots,\ e_{m-1,j_1}s_{j_2} - e_{m-1,j_2}s_{j_1},\ s_{j_2},\ -s_{j_1}\right)$$

is a short vector in the lattice. Consequently, lattice attacks against RLAM with a treated as the key reduce to finding such a short vector. Their cost can be conservatively upper bounded by the cost of solving $\mathrm{LWE}_{mn,2n,q,\mathcal{X}_s,\mathcal{D}_{\sigma_s\sigma_e\sqrt{2n}}}$, where the error distribution $\mathcal{D}_{\sigma_s\sigma_e\sqrt{2n}}$ reflects the products $e_{i,j}\,s_j$ arising in the elimination step and has standard deviations $\sigma_s\sigma_e\sqrt{2n}$.

The concrete security estimates for LARK are summarized in Table 4, with the NTRU-based and RLAM-based components presented separately. These estimates are derived from the attack models described above, taking into account the specific structure of LARK and the information available to the adversary. We report the base-2 logarithms of the estimated computational costs of the best-known classical, quantum, and algebraic attacks under the considered models. Notably, the classical and quantum estimates for the RLAM-based component coincide because BKW [BKW00] is the most efficient attack under the evaluated models, with no quantum speedup reflected in these estimates. We also consider the polynomial-time algebraic attack on LWE proposed by Chen, Ji, and Li in Lemma 6 of [CJL26], which is relevant to RLAM. However, this attack requires $m > 570250$ samples, substantially exceeding the maximum of $m = 512$ samples available within a single LARK iteration.

We emphasize that the sufficient parameter conditions required by the RLWE-to-RLAM reduction in Lemma 5 are not satisfied by the concrete LARK parameters. Nevertheless, these conditions are sufficient for the particular reduction rather than necessary for RLAM hardness, and their violation does not, by itself, imply the existence of an efficient attack. Another existed example is practical RLWR/MLWR based schemes, the theoretical reduction from RLWR/MLWR to LWE requires the stricter conditions than practical instantiations used, such as Lizard, Saber, Spring, and LEAP. The concrete security of LARK is therefore assessed through the cryptanalysis described above, following the established methodology for practical lattice-based cryptography. A separate consideration concerns the reduction losses in Theorem 1. The factors t^2 and d^2 arise from the hybrid arguments over the iterations of LARK.GEN and LARK.PRG, respectively, and must be accounted for when deriving a numerical bound on the distinguishing advantage of LARK. Importantly, the multi-iteration structure is also considered in our concrete security analysis. In particular, the lattice construction described above exploits the algebraic relations between outputs from different LARK.PRG iterations. The corresponding attack-cost estimates therefore account for this source of inter-iteration dependence, rather than treating the outputs as independent instances. Nevertheless, this structural consideration is distinct from the factors t^2 and d^2, which quantify losses in the formal hybrid reduction rather than demonstrated improvements in attack complexity. Since the estimated costs of known attacks do not constitute quantitative upper bounds on the distinguishing advantages of arbitrary adversaries, mechanically subtracting $\log_2(t^2)$ or $\log_2(d^2)$ from these estimates would not yield a justified concrete attack complexity. Accordingly, Table 4 reports the estimated costs of the best-known attacks against the construction-specific

lattice instances considered in our analysis, rather than numerical distinguishing-advantage bounds obtained by directly instantiating Theorem 1.

Table 4: Attack-Cost-Based Security Estimates for LARK

Scheme	NTRU-based Component			RLAM-based Component		
	Classical	Quantum	Algebraic	Classical	Quantum	Algebraic
LARK-128	143.8	138.8	445.0	128.5	128.5	-
LARK-256	280.8	263.4	863.4	268.5	268.5	-

In summary, the exceptional and failure probabilities are analyzed in Section 3.2, while Section 3.3 determines the maximum output lengths supported by the selected parameter sets under their respective target security-level constraints. The formal security reduction of LARK, including all hybrid reduction losses, is presented in Section 4.2. Based on the concrete security analysis in Section 4.3, the minimum estimated attack costs across the evaluated components and attack models are 128.5 bits for LARK-128 and 263.4 bits for LARK-256, representing their estimated concrete security levels under the considered attack models.

4.4 Randomness Test

We evaluated the statistical randomness of LARK using the NIST SP 800-22 test suite [BRS+10], which comprises 15 standard tests for pseudorandom sequences. For each security level, we generated 128 independent samples, each consisting of 10^6 bits, and subjected them to the full battery of tests. All tests were successfully passed at both security levels. In every category, the resulting p-values exceeded the prescribed acceptance thresholds, demonstrating that the output of LARK exhibits strong and consistent statistical properties of randomness.

5 Implementation and Performance

This section presents the implementation details and performance evaluation of LARK. We begin by describing the main optimization techniques, focusing on efficient polynomial multiplication and inversion based on NTT as well as SIMD-oriented vectorized implementations using AVX2 and AVX512 instructions. We then provide comprehensive benchmarking results and analyze the impact of the parameter m and vectorization strategies on the overall efficiency of LARK.

5.1 Polynomial Operation

Polynomial multiplication and inversion dominate the computational cost of the LARK scheme. Hence, optimizing these two operations is crucial for improving overall efficiency. In LARK, both operations are accelerated using Number Theoretic Transform (NTT) techniques. In LARK.GEN at the 128-bit security level, the modulus satisfies $257 \bmod (2 \cdot 512) \neq 1$, which prevents the use of a complete NTT. As a result, a seven-layer incomplete NTT is employed. In this setting, $\mathbf{BaseMul}_Q^N$ uses 4×4 schoolbook polynomial multiplication, while $\mathbf{BaseInv}_Q^N$ performs 4×4 schoolbook polynomial inversion. At the 256-bit security level, where $N = 1024$, the same condition $257 \bmod (2 \cdot 1024) \neq 1$ holds, and an incomplete 7-layer NTT is again adopted. Here, $\mathbf{BaseMul}_Q^N$ relies on 8×8 schoolbook polynomial multiplication. However, directly applying 8×8 schoolbook inversion in $\mathbf{BaseInv}_Q^N$ would be prohibitively expensive. To address this, we follow the approach of [FKPY22, LZL+26] and compute polynomial inverses using a Karatsuba-style even–odd

splitting. Specifically, for $f \in \mathcal{R}_{N,Q}$, we write $f = f_0(x^2) + xf_1(x^2)$, where f_0 and f_1 are polynomials of size $N/2$ defined over $\mathcal{R}_{N/2,Q}$. This decomposition yields

$$\frac{1}{f} = \frac{f_0(x^2) - xf_1(x^2)}{(f_0(x^2) + xf_1(x^2))(f_0(x^2) - xf_1(x^2))} = \frac{f_0(x^2) - xf_1(x^2)}{(f_0^2 - x^2 f_1^2)(x^2)}.$$

Thus, the inversion of f over $\mathcal{R}_{N,Q}$ reduces to computing the inverse of $f_0^2 + xf_1^2$ over a smaller ring. In practice, this transforms the inversion problem from $\mathcal{R}_{1024,257}$ to a smaller ring $\mathcal{R}_{128,257}$, where pointwise inversion can be applied efficiently, resulting in further performance improvements. For LARK.PRG, the modulus satisfies $769 \bmod (2 \cdot 128) = 1$, which enables the use of a complete NTT. Polynomial multiplication is therefore implemented via a full 7-layer NTT, with pointwise multiplication serving as the $\mathbf{BaseMul}_q^n$ operation.

Our NTT implementation combines Montgomery reduction and Barrett reduction to accelerate modular arithmetic. Additionally, we incorporate the lazy reduction technique [Sei18] to decrease the number of modular operations performed during intermediate stages. For each parameter set, the lazy reduction strategy is carefully tuned according to the growth bound of intermediate values during the NTT, which depends on the modulus size and the machine word size. When intermediate values remain within the word-size boundary, modular operations can be safely omitted, resulting in further performance improvements.

5.2 SIMD-Optimized Implementation

LARK is explicitly designed for high implementation efficiency by fully exploiting SIMD parallelism. Most core operations in the scheme, particularly polynomial arithmetic based on the NTT, are well suited for vectorization. In addition to a plain C implementation, we provide optimized implementations using the AVX2 and AVX512 instruction sets, which offer 256-bit and 512-bit SIMD registers, respectively. Under our parameter choices, polynomial coefficients are represented using 16-bit words. Consequently, a single AVX2 register can hold 16 coefficients in parallel, while a single AVX512 register can hold 32 coefficients.

On x86-64 platforms, AVX2 provides 16 architectural YMM registers, while AVX512 provides 32 architectural ZMM registers. During the NTT computation, some registers must be reserved for twiddle factors, temporary products, modular reductions, permutations, and other intermediate values. The remaining registers can be used to store polynomial coefficients and determine the achievable level of parallelism. In our AVX2 implementation, 8 YMM registers remain available for polynomial coefficients. Since each YMM register stores 16 16-bit coefficients, these registers can hold exactly 128 coefficients, allowing one degree-128 polynomial to remain entirely in registers throughout the core NTT computation. In the AVX512 implementation, the larger number of registers provides sufficient capacity to store four degree-128 polynomials simultaneously. Each polynomial occupies four ZMM registers, so four polynomials require 16 ZMM registers, while the remaining registers are sufficient for the auxiliary values required by the computation. This allocation provides a good balance between register pressure and parallelism, enabling four independent NTTs, base multiplications, and inverse NTTs to be processed in parallel without excessive register spilling or intermediate memory traffic. As a result, the implementation reduces load/store overhead, amortizes twiddle-factor and permutation costs across multiple polynomials, and improves SIMD utilization and instruction-level parallelism. The compact polynomial degree is therefore well matched to the available vector registers, and the resulting efficiency gains are evaluated in Section 5.3.

5.3 Performance Benchmark

We implemented LARK in three variants: a plain C version and two vectorized implementations leveraging AVX2 and AVX512 instructions, respectively. All benchmarks were conducted on an Intel Core i9-11900K processor running at a fixed frequency of 3.5 GHz, with TurboBoost and hyperthreading disabled to ensure stable and reproducible measurements. The code was compiled using clang 16.0.6 with flags `-O3 -mtune=native -fomit-frame-pointer`. All results correspond to single-threaded execution.

Tables 5 and 6 present detailed single-thread performance results for LARK at the two security levels. The tables report the CPU cycles per generated byte, the corresponding throughput in Gbps, and the cycle cost of the LARK.GEN algorithm. To obtain a more comprehensive performance evaluation, we vary the parameter m and increase it until optimal performance is achieved on our test platform. The computational cost of the NTT applied to s in Step 4 and sample next s and r in Step 15-18 of Algorithm 6 is amortized over the m output blocks. As m grows, this amortization reduces the per-block cost of the NTT. However, once m becomes sufficiently large, the marginal benefit diminishes, since the NTT cost per output block is already negligible. Beyond this point, further increasing m provides only limited additional speedup. Therefore, the optimal choice of m depends on the specific hardware and software environment.

Table 5: Performance of LARK-128

m	Cycles/byte			Gbps			LARK.GEN (Kcycles)		
	C	AVX2	AVX512	C	AVX2	AVX512	C	AVX2	AVX512
1	17.93	3.84	1.81	1.05	2.41	8.32	54.32	4.59	4.09
8	8.93	1.67	1.28	2.69	11.22	13.05	90.43	12.16	10.97
16	8.72	1.55	1.09	2.88	15.00	18.23	136.03	24.23	17.22
32	8.47	1.54	0.99	3.02	15.88	22.56	227.53	40.17	30.21
48	8.41	1.52	0.96	3.06	16.61	24.47	326.25	56.90	43.00
64	8.37	1.51	0.94	3.08	17.02	25.46	419.80	74.79	55.72
128	8.32	1.47	0.91	3.12	17.54	27.10	790.78	146.81	106.48
256	8.30	1.45	0.91	3.14	17.86	28.01	1527.97	290.24	208.14
512	8.30	1.44	0.90	3.15	18.04	28.49	3031.74	577.58	411.55

Table 6: Performance of LARK-256

m	Cycles/byte			Gbps			LARK.GEN (Kcycles)		
	C	AVX2	AVX512	C	AVX2	AVX512	C	AVX2	AVX512
1	30.97	6.62	2.08	0.60	1.40	7.13	79.38	7.72	6.01
8	9.72	1.82	1.40	2.48	10.30	11.90	116.35	21.79	20.65
16	9.24	1.63	1.12	2.70	13.10	17.45	187.54	47.86	43.51
32	8.92	1.56	1.00	2.86	14.87	22.26	329.20	99.29	87.07
48	8.81	1.54	0.96	2.92	15.75	24.27	467.97	152.80	130.13
64	8.75	1.52	0.94	2.94	16.14	25.31	607.66	204.73	173.67
128	8.70	1.52	0.92	2.98	16.61	27.11	1163.16	413.12	345.73
256	8.66	1.51	0.91	3.00	16.97	28.08	2277.20	826.34	693.83
512	8.64	1.51	0.91	3.01	17.16	28.46	4500.27	1653.82	1394.82

Overall, LARK achieves excellent efficiency at both security levels. The AVX512 implementation reaches as low as 0.90 cycles/byte for LARK-128 and 0.91 cycles/byte for LARK-256, achieving sub-1 cycle/byte performance at both security levels. The corresponding single-core throughput reaches 28.49 Gbps and 28.46 Gbps, respectively. Compared with the plain C implementation, the AVX2 and AVX512 implementations

achieve speedups of up to 5.76× and 14.89×, respectively.

6 Conclusion

In this work, we present LARK, a lattice-based pseudorandom number generator that combines compact keys with high performance. At the theoretical level, we formalize the Ring Learning with Alternating Moduli (RLAM) problem as the ring analogue of Learning with Alternating Moduli problem and establish its hardness via a reduction from Ring Learning with Errors. In the design of LARK, the RLAM formulation allows us to incorporate the high-order information discarded in LEAP into the security estimation.

At the construction level, this additional security contribution enables LARK to achieve the target security levels with smaller parameters. We further introduce an efficient masked-rejection mechanism to eliminate the bias introduced by modulus reduction over coprime moduli. Together, these design choices lead to substantially reduced key sizes and improved implementation efficiency. Our experimental results show that LARK consistently outperforms previous lattice-based PRGs while significantly reducing key sizes, with the AVX512 implementation achieving sub-1 cycle/byte performance at both the 128-bit and 256-bit security levels. Overall, LARK demonstrates that the RLAM viewpoint provides an effective foundation for combining compact parameters with high-throughput lattice-based pseudorandom generation.

References

[ADPS16] Erdem Alkım, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. In *USENIX Security Symposium*, 2016.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with Rounding, Revisited. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 57–74, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[ASA16] Jacob Alperin-Sheriff and Daniel Apon. Dimension-Preserving Reductions from LWE to LWR. *IACR Cryptol. ePrint Arch.*, 2016:589, 2016.

[ASP13a] Jacob Alperin-Sheriff and Chris Peikert. Practical Bootstrapping in Quasilinear Time. Cryptology ePrint Archive, Paper 2013/372, 2013.

[ASP13b] Jacob Alperin-Sheriff and Chris Peikert. Practical Bootstrapping in Quasilinear Time. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 1–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BBL+15] Abhishek Banerjee, Hai Brenner, Gaëtan Leurent, Chris Peikert, and Alon Rosen. SPRING: Fast pseudorandom functions from rounded ring products. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption*, pages 38–57, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BDFK17] Charles Bouillaguet, Claire Delaplace, Pierre-Alain Fouque, and Paul Kirchner. Fast Lattice-Based Encryption: Stretching SPRING. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 125–142, Cham, 2017. Springer International Publishing.

[BDH+23] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. TurboSHAKE. Cryptology ePrint Archive, Paper 2023/342, 2023.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[BGM+16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the Hardness of Learning with Rounding over Small Modulus. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 209–224, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring Crypto Dark Matter: New Simple PRF Candidates and Their Applications. Cryptology ePrint Archive, Paper 2018/1218, 2018.

[BK15] Elaine B. Barker and John M. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators:, 2015-06-01 04:06:00 2015.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model, 2000.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 112–117, 1982.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom Functions and Lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 719–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BRS+10] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, 2010-09-16 2010.

[CJL26] Yilei Chen, Liheng Ji, and Wenjie Li. Learning with Alternating Moduli, Arora-Ge over Composite Moduli, and Weak PRFs. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026*, pages 201–231, Cham, 2026. Springer Nature Switzerland.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient Identity-Based Encryption over NTRU Lattices. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 22–41, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[FHK+19] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU. 2019.

[FKPY22] Pierre-Alain Fouque, Paul Kirchner, Thomas Pornin, and Yang Yu. BAT: Small and Fast KEM over NTRU Lattices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(2):240–265, Feb. 2022.

[FSK15] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. Generating randomness. *Cryptography Engineering: Design Principles and Practical Applications*, pages 135–161, 2015.

[HLR+26] Kai Hu, Gregor Leander, Håvard Raddum, Arne Sandrib, and Aleksei Udovenko. Cryptanalysis of Two Alternating Moduli Weak PRFs. Cryptology ePrint Archive, Paper 2026/482, 2026.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[KSF99] John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In *International Workshop on Selected Areas in Cryptography*, pages 13–33. Springer, 1999.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LW20] Feng-Hao Liu and Zhedong Wang. Rounding in the Rings. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 296–326, Cham, 2020. Springer International Publishing.

[LZL+26] Yijian Liu, Yu Zhang, Xianhui Lu, Yao Cheng, and Yongjian Yin. DAWN: Smaller and Faster NTRU Encryption via Double Encoding. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025*, pages 396–427, Singapore, 2026. Springer Nature Singapore.

[Nat24a] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication FIPS 204, National Institute of Standards and Technology, 2024.

[Nat24b] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication FIPS 203, National Institute of Standards and Technology, 2024.

[oSND15] National Institute of Standards, Technology (NIST), and Morris J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015-08-04 00:08:00 2015.

[PMS21] Alice Pellet-Mary and Damien Stehlé. On the Hardness of the NTRU Problem. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 3–35, Cham, 2021. Springer International Publishing.

[Pol71] John M Pollard. The fast Fourier transform in a finite field. *Mathematics of computation*, 25(114):365–374, 1971.

[PRSD17] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 461–473, New York, NY, USA, 2017. Association for Computing Machinery.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography, 2018. Report 2018/039.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as Secure as Worst-Case Problems over Ideal Lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 27–47, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient Public Key Encryption Based on Ideal Lattices. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 617–635, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Yao82] Andrew Chi-Chih Yao. Theory and Applications of Trapdoor Functions (Extended Abstract). In *IEEE Annual Symposium on Foundations of Computer Science*, 1982.

[ZLL+25] Yu Zhang, Xianhui Lu, Yijian Liu, Yongjian Yin, and Kunpeng Wang. LEAP: High-Performance Lattice-Based Pseudorandom Number Generator. *IACR Transactions on Symmetric Cryptology*, 2025(3):151–182, Sep. 2025.

A Hardness of Decisional Ring-LAM from Decisional Ring-LWE

The proof of Lemma 5 adapts the reduction of Chen et al. [CJL26, Theorem 24] to the ring setting. Given an RLWE sample over $\mathcal{R}_{q,n}$, the reduction multiplies both components by p modulo q and then applies the coefficientwise modulus-reduction map $[\cdot]_{q\to p}$. The additional step in the ring setting is to use the condition $\gcd(q, s_0, \ldots, s_{n-1}) = 1$, which ensures that, for uniform $a \leftarrow U(\mathcal{R}_{q,n})$, each coefficient of as is uniformly distributed over $\mathbb{Z}_q$. A union bound over the mn coefficients then bounds the probability that modulus wraparound changes the reduced output.

Proof. Let $\mathcal{A}$ be any PPT distinguisher against $\mathrm{DRLAM}_{m,n,q,p,\mathcal{X}_s}$. We construct a PPT distinguisher $\mathcal{B}$ against $\mathrm{DRLWE}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}$.

The input to $\mathcal{B}$ consists of m samples

$$\big((a_0, b_0), \ldots, (a_{m-1}, b_{m-1})\big) \in (\mathcal{R}_{q,n} \times \mathcal{R}_{q,n})^m,$$

where either $b_i = a_i s + e_i \pmod q$ for a common secret $s \leftarrow \mathcal{X}_s$ and independently sampled errors $e_i \leftarrow \mathcal{X}_e$, or $b_0, \ldots, b_{m-1}$ are independently and uniformly sampled from $\mathcal{R}_{q,n}$.

For every $i \in \{0, \ldots, m-1\}$, algorithm $\mathcal{B}$ computes

$$a'_i := pa_i \pmod q, \qquad b'_i := pb_i \pmod q.$$

It then gives

$$\big((a'_0, [b'_0]_{q\to p}), \ldots, (a'_{m-1}, [b'_{m-1}]_{q\to p})\big)$$

to $\mathcal{A}$ and forwards the output bit of $\mathcal{A}$.

Since $\gcd(p, q) = 1$, multiplication by p is a permutation of $\mathbb{Z}_q$, and hence also a permutation of $\mathcal{R}_{q,n}$.

- Case 0 (uniform case). Suppose that $b_0, \ldots, b_{m-1}$ are independently and uniformly sampled from $\mathcal{R}_{q,n}$. Then $b'_i = pb_i \bmod q$ is also uniform in $\mathcal{R}_{q,n}$ for every i. Similarly, $a'_i = pa_i \bmod q$ is uniform in $\mathcal{R}_{q,n}$. Therefore, the input given to $\mathcal{A}$ is distributed identically to
$$\big((a'_0, [u_0]_{q\to p}), \ldots, (a'_{m-1}, [u_{m-1}]_{q\to p})\big),$$
where all $a'_i, u_i \leftarrow U(\mathcal{R}_{q,n})$ independently. This is exactly a decisional RLAM no-instance.

- Case 1 (RLWE case). Suppose that $b_i = a_i s + e_i \pmod q$ for $s \leftarrow \mathcal{X}_s$ and $e_i \leftarrow \mathcal{X}_e$. Then
$$\begin{aligned} b'_i &= pb_i \pmod q \\ &= p(a_i s + e_i) \pmod q \\ &= (pa_i)s + pe_i \pmod q \\ &= a'_i s + pe_i \pmod q. \end{aligned}$$

For every i, let $r_i := a'_i s \pmod q$, and write $r_i = \sum_{j=0}^{n-1} r_{i,j} X^j$, where each $r_{i,j}$ is represented by its canonical representative in $\{0, \ldots, q-1\}$. Define the events

$$\begin{aligned} E_e &:= \{\|e_i\|_\infty \le B \text{ for every } i \in \{0, \ldots, m-1\}\}, \\ E_r &:= \{r_{i,j} \in [pB, q-pB) \text{ for every } i \in \{0, \ldots, m-1\},\ j \in \{0, \ldots, n-1\}\}, \end{aligned}$$

and let $G := E_e \cap E_r$.

Whenever G occurs, for every i and j we have

$$0 \le r_{i,j} + pe_{i,j} < q.$$

Thus, reduction modulo q does not wrap around, and

$$\begin{aligned} [r_{i,j} + pe_{i,j}]_{q\to p} &= (r_{i,j} + pe_{i,j}) \bmod p \\ &= r_{i,j} \bmod p \\ &= [r_{i,j}]_{q\to p}. \end{aligned}$$

Applying this equality coefficientwise gives

$$[b'_i]_{q\to p} = [a'_i s + pe_i]_{q\to p} = [a'_i s]_{q\to p}$$

for every i. Therefore, whenever G occurs, the input given to $\mathcal{A}$ is equal to

$$\big((a'_0, [a'_0 s]_{q\to p}), \ldots, (a'_{m-1}, [a'_{m-1} s]_{q\to p})\big),$$

which is a decisional RLAM yes-instance.

We next bound the probability that G does not occur. First, by a union bound over the m independently sampled error polynomials,

$$\begin{aligned} \Pr[\neg E_e] &= \Pr\left[\exists i : \|e_i\|_\infty > B\right] \\ &\le m \cdot \Pr_{e\leftarrow\mathcal{X}_e}\left[\|e\|_\infty > B\right]. \end{aligned}$$

To bound $\Pr[\neg E_r]$, fix any $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, n-1\}$, and $s \in \mathrm{Supp}(\mathcal{X}_s) \subseteq \mathcal{R}^\dagger_{q,n}$. The coefficient $r_{i,j}$ of $a'_i s$ is a linear form in the coefficients of a'_i, and the coefficient vector of this linear form is a signed permutation of $(s_0, \ldots, s_{n-1})$. By the assumption $\gcd(q, s_0, \ldots, s_{n-1}) = 1$, this linear form is surjective onto $\mathbb{Z}_q$. Since $a'_i \leftarrow U(\mathcal{R}_{q,n})$, it follows that $r_{i,j} \leftarrow U(\mathbb{Z}_q)$. Consequently,

$$\begin{aligned} \Pr\left[r_{i,j} \notin [pB, q-pB) \,|\, s\right] &= \Pr\left[r_{i,j} \in [0, pB) \cup [q-pB, q) \,|\, s\right] \\ &= \frac{2pB}{q}. \end{aligned}$$

This bound holds for every $s \in \mathrm{Supp}(\mathcal{X}_s) \subseteq \mathcal{R}^{\dagger}_{q,n}$, and therefore also without conditioning on s. A union bound over the mn coefficient positions yields

$$\begin{aligned}\Pr[\neg E_r] &\leq \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} \Pr\left[r_{i,j} \notin [pB, q-pB)\right] \\ &\leq \frac{2mnpB}{q}.\end{aligned}$$

Hence,

$$\begin{aligned}\Pr[\neg G] &\leq \Pr[\neg E_e] + \Pr[\neg E_r] \\ &\leq m \cdot \Pr_{e \leftarrow \mathcal{X}_e}\left[\|e\|_\infty > B\right] + \frac{2mnpB}{q} \\ &= \delta.\end{aligned}$$

By assumption,

$$\frac{2mnpB}{q} < \frac{1}{T},$$

and therefore $\delta = \mathsf{negl}(n)$.

It remains to relate the distinguishing advantages. Define

$$\begin{aligned}\mathrm{Acc}_{\mathsf{yes}} &:= \Pr\left[\mathcal{A}\left((a_0, [a_0 s]_{q\to p}), \ldots, (a_{m-1}, [a_{m-1}s]_{q\to p})\right) = 1\right], \\ \mathrm{Acc}_{\mathsf{no}} &:= \Pr\left[\mathcal{A}\left((a_0, [u_0]_{q\to p}), \ldots, (a_{m-1}, [u_{m-1}]_{q\to p})\right) = 1\right],\end{aligned}$$

where $a_0, \ldots, a_{m-1}, u_0, \ldots, u_{m-1} \leftarrow U(\mathcal{R}_{q,n})$ independently and $s \leftarrow \mathcal{X}_s$. By definition,

$$\mathrm{Adv}^{\mathrm{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{A}) = \left|\mathrm{Acc}_{\mathsf{yes}} - \mathrm{Acc}_{\mathsf{no}}\right|.$$

In the uniform case, $\mathcal{B}$ gives $\mathcal{A}$ an exact decisional RLAM no-instance. Hence,

$$\Pr[\mathcal{B}(\mathsf{uniform}) = 1] = \mathrm{Acc}_{\mathsf{no}}.$$

In the RLWE case, consider the input generated by $\mathcal{B}$ and the corresponding RLAM yes-instance generated using the same $a'_0, \ldots, a'_{m-1}$, the same secret s, and the same internal randomness of $\mathcal{A}$. The two inputs are identical whenever G occurs. Thus, the output bits can differ only when $\neg G$, and therefore

$$\left|\Pr[\mathcal{B}(\mathsf{RLWE}) = 1] - \mathrm{Acc}_{\mathsf{yes}}\right| \leq \Pr[\neg G] \leq \delta.$$

Using the reverse triangle inequality, we obtain

$$\begin{aligned}\mathrm{Adv}^{\mathrm{DRLWE}}_{m,n,q,\mathcal{X}_s,\mathcal{X}_e}(\mathcal{B}) &= \left|\Pr[\mathcal{B}(\mathsf{RLWE}) = 1] - \Pr[\mathcal{B}(\mathsf{uniform}) = 1]\right| \\ &= \left|\Pr[\mathcal{B}(\mathsf{RLWE}) = 1] - \mathrm{Acc}_{\mathsf{no}}\right| \\ &\geq \left|\mathrm{Acc}_{\mathsf{yes}} - \mathrm{Acc}_{\mathsf{no}}\right| - \left|\Pr[\mathcal{B}(\mathsf{RLWE}) = 1] - \mathrm{Acc}_{\mathsf{yes}}\right| \\ &\geq \mathrm{Adv}^{\mathrm{DRLAM}}_{m,n,q,p,\mathcal{X}_s}(\mathcal{A}) - \delta.\end{aligned}$$

Finally, $\mathcal{B}$ performs only $2m$ scalar multiplications in $\mathcal{R}_{q,n}$ and mn coefficientwise modulus reductions in addition to running $\mathcal{A}$. Therefore,

$$\mathsf{Time}(\mathcal{B}) = \mathsf{Time}(\mathcal{A}) + \mathrm{poly}(m, n, \log q).$$

This completes the proof. □