

ARCANE-Octarine

Algorithm Specifications and Supporting Documentation

This submission is from the ARCANE team, listed in alphabetical order:

- Jose Maria Bermudo Mera, PQShield, Belgium
- Joppe W. Bos, NXP Semiconductors, Belgium
- Angshuman Karmakar, Indian Institute of Technology Kanpur, India
- Joost Renes, NXP Semiconductors, United States of America
- Sujoy Sinha Roy, Graz University of Technology, Austria
- Michiel Van Beirendonck, Belfort Labs, Belgium
- Frederik Vercauteren, KU Leuven, Belgium
- Peng Wang, School of Cryptology, University of Chinese Academy of Sciences, China
- Yuewu Wang, School of Cryptology, University of Chinese Academy of Sciences, China
- Fangyu Zheng, School of Cryptology, University of Chinese Academy of Sciences, China

Contact: Peng Wang

Email: `p-wang@ucas.ac.cn`

Website: <https://soc.ucas.ac.cn/index.php/zh/ARCANE>
<https://www.pq-arcane.org>

Contents

1	The ARCANE suite	1
2	Introduction	4
2.1	Overview of the Basic Approach	4
2.2	Relation to Existing Work	5
2.3	Security	7
3	Preliminaries	8
3.1	Notation	8
3.2	Lattice-based Cryptography	9
3.2.1	Learning with Errors	9
3.2.2	Learning with Rounding	10
3.2.3	Security of Learning with Rounding	11
3.3	Rings as Structured Modules	12
3.4	Radical Ring-LWR	13
4	Algorithm Description	14
5	Design Rationale	25
5.1	Criterion for Parameter Selection	25
5.2	Correctness and Rejection-Probability Analysis	25
5.3	Other Considerations	26
6	Security Statements and Analyses	27
6.1	Theoretical Security	27
6.1.1	RR-LWR hard problem	27
6.1.2	Unforgeability of Octarine	30
6.1.3	Beyond-unforgeability properties	34
6.2	Practical Security	35
6.2.1	Physical Security	35
7	Performance Evaluation	36
7.1	Arithmetic Implementations	36
7.2	Performance Analyses	37
7.3	Performance Test	39
A	SageMath Script	42
A.1	Security estimates	42
B	Checking List	45

1 The ARCANE suite

Post-quantum cryptography requires algorithms that are secure against quantum adversaries and allow replacing traditional schemes like RSA and ECC for use cases like key exchange (or encapsulation) and digital signatures. The ARCANE suite introduces two algorithms:

- ARCANE-Mithril: a post-quantum Key Encapsulation Mechanism (KEM);
- ARCANE-Octarine: a post-quantum Digital Signature (DS) scheme.

We highlight the main features that are shared across **Mithril** and **Octarine**.

Innovativeness. The security is based on the *Radical Ring Learning-With-Rounding* (RR-LWR) problem [BRVW26], a Ring-LWR variant over a radical number field. In this variant, rounding is applied coefficient-wise in the transported monomial basis, i.e., the image of the radical-ring monomial basis under the number-field isomorphism formalized in Theorem 3. It can be used as an alternative for Module Learning-With-Rounding (Module-LWR). It inherits the benefits from Module-LWR (flexible parameter selection) while retaining most of the performance advantages from Ring-LWR. Both the KEM and DS scheme operate over a radical ring of the form

$$\mathbf{S}_{q,n,k} = (\mathbb{Z}_q[y]/(y^n + 1))[x]/(x^k - y - 2), \quad (1)$$

where q and n are powers of two, and k is an arbitrary positive integer. There exists a straight-forward isomorphism to the number-field representation $\mathbf{S}_{q,n,k} \cong \mathbb{Z}_q[x]/((x^k - 2)^n + 1)$ [BRVW26, Section 4], giving the radical-ring representation used for our transported-basis Ring-LWR variant. The ring $\mathbf{S}_{q,n,k}$ is a degree k *radical* extension of the $2n$ -th cyclotomic field. As such, standard implementation techniques (such as NTTs or other polynomial multiplication routines) can be used if desired, while the radical extension allows more efficient matrix generation than a module structure.

We choose a power of two q instead of a prime. This makes **Mithril** similar in design to Saber [DKR⁺20, Section A.2], but with a different choice of ideal lattice (RR-LWR instead of Module-LWR). We also define a digital signature scheme **Octarine** following the well-established Fiat-Shamir with Aborts paradigm [Lyu09, Lyu12] adapted to the RR-LWR setting. A previous LWR-based signature scheme, MLWRSign, was proposed by Okada et al. [OTF⁺20]. We choose a different approach by defining $\mathbf{s}_2$ as the deterministic error from the public key $\mathbf{s}_2 = \mathbf{A}\mathbf{s}_1 - \frac{q}{p} \lfloor \frac{p}{q} \mathbf{A}\mathbf{s}_1 \rfloor$, rather than the rational quantity $\lfloor \frac{p}{q} \mathbf{A}\mathbf{s}_1 \rfloor - \frac{p}{q} \mathbf{A}\mathbf{s}_1$ in MLWRSign. Moreover, we choose to compute $\mathbf{s}_2$ during key generation instead of during signature generation. This simplifies the design and makes the overall construction closer to CRYSTALS-Dilithium [DKL⁺18] than MLWRSign.

Simplicity. We use a Ring-LWR variant over the radical number field with coefficient-wise rounding in the transported monomial basis described above; the concrete estimates account for the resulting basis distortion relative to the usual canonical-embedding viewpoint. The resulting KEM and digital signature scheme target conservative security notions such as IND-CCA2 [BDPR98] for the KEM and, for the digital signature scheme, SUF-CMA [ADR02] in the random oracle model (ROM).

We aim to simplify existing designs in various ways. Firstly, by relying on a (radical) ring structure instead of a module structure. Secondly, by minimizing the complexities of various parameter choices through the use of a power-of-two q . All sampling is done in *uniform* distributions in intervals $[-\alpha, \alpha - 1]$ modulo a power of two α , and we unify the choice of α across the parameter sets as much as possible. Generation of random elements in $[-\alpha, \alpha - 1]$ is done by simply generating $\log_2(2\alpha)$ random bits. This implies that no rejection sampling is required during secret, public-matrix, and masking-vector coefficient sampling and no (pseudo-)randomly

generated bits need to be discarded. We select parameters such that failure probabilities are always negligibly small. In particular for the digital signature scheme, parameters for decomposition and the Fiat-Shamir abort rejection loop are all based on powers of two, making bound checks easy by simply checking if a carry occurred. We also reduce the number of rejections during signature generation to reduce the variance in the signing time, making adoption in application protocols potentially easier. For example, **Octarine-128** on average requires 2.13 rounds to complete a signing operation, with a standard deviation of 1.55. For comparison, **Dilithium2** has 4.25 rounds on average with a standard deviation of 3.72.

Flexibility. We inherit some of the advantages of lattice-based cryptosystems. Selecting q to be a 13-bit or 24-bit power of two allows values modulo q to fit within most modern-day CPU words, making implementation straightforwardly possible. We make several design decisions that increase the potential for secure and flexible implementation across architectures:

- While LWR with a prime modulus q can be advantageous for fast implementation, through the use of asymptotically efficient NTTs, a power-of-two q has been shown to be preferred for masking [CJRR99, PR13]. Masking is often required for protecting against side-channel and fault attacks, which is especially important in the setting of embedded systems and high-assurance implementations. Arithmetic-to-Boolean conversions (and vice-versa) [CGV14, BBE⁺18], a frequently used tool to create masked implementations, are less complex for powers of two, as are sampling routines that do not require rejection sampling. As embedded systems are already naturally constrained, simplifying side-channel and fault protection countermeasures will help reduce the cost of implementations.
- By using a radical ring we reduce the size of the public matrix, which directly impacts the memory requirements. For example, the matrix size is reduced by almost a factor of seven compared to the ML-DSA-87 matrix: this matrix requires over 57.3 kB if 32 bits per coefficient are used or over 41.2 kB when stored in packed form. In comparison, we can represent our public matrix in about 8.2 kB if 32 bits per coefficient are used, or 6.1 kB in packed form. This not only reduces the cost to generate the matrix, but also reduces the memory impact to store the matrix across multiple uses (in particular, for signature generation). This gives rise to better time-memory trade-offs for constrained systems.
- We represent all variables in standard domains to avoid dependencies on specific implementation or algorithmic techniques in the description of the algorithms. While performance advantages might be achieved by representing elements in specific ways, for example in NTT domains, this can limit the portability of the algorithms when targeting different platforms. The arithmetic for the ARCANE suite algorithms can be efficiently implemented using NTTs [CT65, Pol71], Toom-Cook [Too63, Coo66], Karatsuba [KO62], or with the use of existing hardware by using Kronecker substitution [Kro82, BRvV22]. We show that performance remains competitive with respect to state-of-the-art lattice-based cryptosystems. In particular, by representing the Mithril private key polynomial $\mathbf{s}$ in normal domain instead of NTT domain, we only require $\log_2(2\eta)$ bits per coefficient, as opposed to $\log_2(q)$ bits for Kyber, significantly reducing its size.

Compatibility. The security level of this RR-LWR family is largely determined by the lattice dimension, which depends on the polynomial degree (n) and the radical extension degree (k). By increasing the power-of-two n we improve the overall efficiency of the arithmetic operations, while varying k offers more fine-grained control over the security level akin to Module-LWR. While $n = 256$ is a common choice for KEMs, it leads to challenges for achieving a 128-bit security level as the lattice dimension is selected as either 512 or 768. As dimension 512 can provide insufficient security (or leave very little margin) and dimension 768 generally provides

a security level that is too high, we take a conservative design approach for the KEM and select $n = 128$ with $k = 5$ to work with a lattice of dimension 640. Concretely, this increases the bit security from 140 for Kyber and Saber to 153 bits under the refined attack model. The ideal lattice based on our radical extension can be represented with an element that is *linear* in k , as opposed to quadratic in k for Module-LWE/LWR, so we mitigate some of the impact on the cost of matrix generation. Our design strategy results in a slight increase of the key and ciphertext size and impacts the overall performance compared to using a smaller lattice. However, we believe this to be acceptable to ensure one has more confidence in the achieved 128-bit security level.

We unify $n = 128$ across all KEM parameter sets to simplify the implementation in both software and hardware. For the digital signature we select $n = 1024$ across all parameter sets to maximize the performance based on the cyclotomic sub-ring structure, varying $k = 1, 2, 5$. There are many more parameters to play with, affecting performance, key sizes, rejection rates, signing time variance, etc. We select parameters such that the public key and signature remain similar to other lattice-based proposals, while improving performance and reducing the variance of the signature generation for some of the common parameter sets: e.g., the number of expected repetitions in the signature generation is reduced by a factor two when comparing ARCANE-Octarine-128 versus CRYSTALS-Dilithium-2 [DKL⁺18] (2.1 versus 4.3 repetitions, respectively).

To maximize the re-use between the digital signature and the KEM, we select all sampling distributions to be uniform distributions in intervals $[-\alpha, \alpha - 1]$ modulo a power of two α . For both the digital signature scheme and the KEM we select $\alpha = 2$ as the default sampling distribution, deviating only for one of the parameter sets as the security impact would be too significant. Packing and rounding operations based on powers of two are efficient and optimally use the required bits. In both cases we select n to be an NTT-friendly degree (i.e., a power of two), allowing arithmetic based on NTTs if this is the optimal choice for a given platform. While n is not the same between the KEM and digital signature, a shared implementation can be achieved by using the multiplication with $n = 128$ as a sub-routine for a degree-1024 multiplication. For example, with $q = 2^{13}$ for the KEM and $q = 2^{24}$ for the digital signature, a 32-bit NTT-friendly prime p_1 can be selected for using NTTs, which can be used for the KEM, as well as for the digital signature using the ring $\mathbb{Z}_{p_1 p_2} \cong \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2}$. This can be implemented with a single module to operate in $\mathbb{Z}_p$ for an arbitrary prime p , re-using most of the code, with a small layer for a 2-prime CRT computation on top to convert back from $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2}$ to $\mathbb{Z}_p$. We demonstrate this in our reference implementation and optimized implementations, highlighting its efficiency.

Note that the choice of prime, or the use of NTT at all, is not part of the specification and is purely an implementation choice, and other implementation methods can be chosen if they result in better performance.

Extensibility. The constructions are intended to serve as reusable building blocks beyond the concrete instances specified in this document. The KEM can be integrated into higher-level protocols for ephemeral key establishment, KEM-DEM encryption, and authenticated key-establishment mechanisms, provided that the surrounding protocol supplies the required authentication, key-confirmation, and composition guarantees. It can also be combined with classical or other post-quantum key-establishment mechanisms in a *hybrid* mode, supporting algorithm agility during migration.

At the suite level, the modular organization of the schemes also leaves room for future variants. In particular, alternative parameter profiles, hash/XOF/KDF instantiations, or implementation profiles may be considered without changing the underlying radical-ring operations. Such variants should nevertheless be accompanied by updated correctness bounds, concrete security estimates, and implementation validation.

Octarine supports standard signing applications such as general-purpose message signing,

code signing, digital certificates, and secure attestation. Moreover, because the scheme follows the Fiat–Shamir paradigm, the same algebraic setting may provide a starting point for more advanced Fiat–Shamir-based signing constructions, including threshold signatures [PN25].

Thus, the RR-LWR framework should be viewed not only as a collection of fixed KEM and signature instances, but also as a platform for future protocol integration and carefully analyzed variants that reuse the same radical-ring structure.

Performance Advantages on Various Platforms. The conservative design choices of the ARCANE suite allow implementations to be optimized for a wide range of target platforms. In particular, the specification deliberately decouples the choice of polynomial-arithmetic method from the algorithm: implementations may use NTT-based, Toom–Cook, Karatsuba, schoolbook multiplication, or architecture-specific techniques when appropriate.

For meaningful comparisons with established lattice-based schemes, the performance sections additionally report SHAKE-based benchmarking configurations where applicable, alongside the submission measurements based on the specified `pseudoXOF` instantiation. In the KEM setting, the SHAKE-based AVX2 comparison covers Mithril-128 and Mithril-256; the optimized AVX2 implementation of Mithril-128 is slower than ML-KEM-512 by a factor of about 1.4. This difference can be partly attributed to the larger estimated security margin of Mithril-128 compared with ML-KEM-512. At the higher security level, the performance gap between Mithril-256 and ML-KEM-1024 decreases to about a factor of 1.1, again with Mithril providing a larger estimated security margin.

In the digital-signature setting, Octarine-128 is faster than ML-DSA-44 for key generation and signing, while ML-DSA-44 remains faster for verification. At the higher security level, Octarine-256 outperforms ML-DSA-87 across the measured signing and verification operations, with speedups of 1.13 for signing and 1.23 for verification. These results show that the ARCANE design achieves competitive performance while retaining conservative security margins and leaving substantial flexibility for platform-specific optimization.

We note that the advantages of the ARCANE schemes, like reduced memory consumption and pseudo-randomness generation for the public key matrix, will be increasingly beneficial for smaller Arm or RISC-V based platform cores where the cost of the XOF dominates. Moreover, the simple design choices should significantly help with side-channel and fault-protected implementations, as demonstrated for example by the advantages of Saber over Kyber, as well as remaining flexible for hardware designs that can be based on various types of multipliers. We aim to quantify these improvements in future versions of the specification.

2 Introduction

For the remainder of this document we focus on the specification of the digital signature scheme ARCANE-Octarine (or the shorter Octarine).

2.1 Overview of the Basic Approach

The Fiat–Shamir with aborts paradigm was originally proposed in [Lyu09] and forms the basis of several practical lattice signature schemes. We give a high-level description of how the approach works in our setting. A detailed algorithm specification is available in Algorithm 3.

Key generation. The signer samples a public matrix $\mathbf{A}$ from a seed ρ and draws a secret vector $\mathbf{s}_1$ with bounded coefficients. The LWR rounding $\mathbf{b} = \lfloor \frac{p}{q} \mathbf{A} \mathbf{s}_1 \rfloor$ forms the main component of the public key. The rounding error $\mathbf{s}_2 = \mathbf{A} \mathbf{s}_1 - \frac{q}{p} \mathbf{b}$ is stored in the secret key. Intuitively, $\mathbf{s}_2$ records the discrepancy introduced by LWR rounding. During signing, it allows the signer to

compensate for this discrepancy when forming the verifier’s reconstructed commitment, while avoiding rational arithmetic. See Section 5 for the motivation behind this definition.

Dilithium [DKL⁺18] (which became the ML-DSA standard) introduced a public-key compression method using the Power2Round function and including a hint vector in the signature. We also use this technique to reduce the size of the signature further by an additional d bits, where $d \in \{11, 13, 14\}$ for the different parameter sets.

Signing. The signer draws a masking vector $\mathbf{y}$ with uniformly random coefficients bounded by γ_1 , computes the commitment $\mathbf{w} = \mathbf{A}\mathbf{y}$, and extracts its high bits $\mathbf{w}_1 = \lfloor \mathbf{w}/(2\gamma_2) \rfloor$ (where γ_2 is a power of two, hence the division corresponds to a simple right bit shift). A challenge seed $\tilde{c}$ is derived by hashing the message digest μ together with $\mathbf{w}_1$; the challenge polynomial $c = \text{SampleInBall}(\tilde{c})$ is computed, and a candidate response $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ is formed. To prevent $\mathbf{z}$ from leaking information about the secret key, the signer applies *rejection sampling*. The first rejection test bounds $\|\mathbf{z}\|_\infty$, ensuring that the distribution of accepted responses is sufficiently independent of $\mathbf{s}_1$. The second rejection test bounds the low-order part of $\mathbf{w} + c\mathbf{s}_2$, ensuring that the verifier can recover the same high-order commitment using the public key and the hint bits. The parameters are chosen so that the expected number of repetitions is small: 2.1, 2.3, and 3.6 expected repetitions for Octarine-128, Octarine-256, and Octarine-512, respectively.

Verification. Verification recomputes an approximation to the original commitment from the response $\mathbf{z}$, the challenge c , and the compressed public key

$$\mathbf{w}'_{\text{Approx}} = \mathbf{A}\mathbf{z} - c \cdot \frac{q}{p}\mathbf{b}_1 \cdot 2^d,$$

where $\mathbf{b}_1$ is the high-order part of the public key, d is the number of bits dropped in the Power2Round compression, and c is recovered from the challenge hash $\tilde{c}$. This value differs from the signer’s commitment only by the controlled rounding terms from public-key compression and LWR rounding. The hint vector $\mathbf{h}$ is used to correct for the carries caused by the missing low-order bits $\mathbf{b}_0$, yielding $\mathbf{w}'_1$. The signature is accepted if $\mathbf{w}'_1 = \mathbf{w}_1$, i.e., $\tilde{c} = H(\mu \parallel \mathbf{w}'_1)$, and $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$.

Our parameters target three security levels and are selected conservatively with respect to the best known classical and quantum attacks against the underlying RR-LWR and SIS-type hardness assumptions. The scheme is modular in the sense that adjusting the ring dimension scales most of the security level without changing the overall structure of the algorithms, though careful selection of other parameters (mainly p , η , γ_1 , γ_2 and τ) is still required to balance the performance characteristics with the exact security level.

2.2 Relation to Existing Work

We compare Octarine to the most relevant existing lattice-based digital signature schemes. Among existing proposals, the closest points of reference are CRYSTALS-Dilithium [DKL⁺18] (standardized as ML-DSA) and the LWR-based signature scheme MLWRSign [OTF⁺20]. Other digital signature schemes of note are Falcon [PFH⁺22] and HAETAE [CCD⁺24]. As these schemes differ more substantially from ours, we focus on performance and size when comparing against them.

Our construction follows the well-established Fiat–Shamir with aborts paradigm, but combines it with a different algebraic setting and a different treatment of the rounding error. The resulting design is intended to improve the overall balance between conceptual simplicity, implementation efficiency, memory usage, and robustness of hardened implementations. In particular, our goal is not only to obtain competitive asymptotic security and performance, but also to reduce the engineering complexity of deploying the scheme in constrained and side-channel-relevant environments.

CRYSTALS-Dilithium / ML-DSA. Octarine shares the high-level signing structure of ML-DSA, but departs from it in several important design choices.

First, we instantiate the scheme over the radical ring $\mathbf{S}_{q,n,k}$ and base security on the **RR-LWR** assumption rather than on Module-LWE. This choice gives rise to a different implementation profile. In particular, we select q as a power of two. This choice does not preclude the use of fast multiplication techniques, including NTT-based implementations, but it avoids making the design dependent on a specific NTT-friendly prime modulus. In our setting, power-of-two arithmetic leads to a simpler and more uniform algorithmic structure, while still allowing efficient arithmetic through NTTs, Toom–Cook, Karatsuba, or other multiplication techniques depending on the target platform. From a specification perspective, this removes a substantial amount of complexity and makes the construction less tied to one particular implementation strategy. While this has an impact on performance, we show in Section 7 that **Octarine-128** is faster than ML-DSA-44 in key generation and signing, but ML-DSA-44 is faster in verification, whereas **Octarine-256** is faster than ML-DSA-87 in all measured operations.

Second, our parameter choices are driven by a conservative security objective across all targeted security levels. In particular, we include a 512-bit target security level and adapt the hash domain and output lengths accordingly. For uniformity at the 512-bit security level, we enlarge the relevant hash outputs (e.g., the digests tr and μ and the per-signature seed ρ'' to 1024 bits), while the challenge hash $\tilde{c}$ retains its security-level-dependent length $\lambda/4$; smaller outputs would suffice at the lower security levels. This avoids level-dependent special cases in the specification and implementation, and keeps the overall structure of the scheme unchanged across parameter sets. More generally, our 128-bit and 256-bit parameter sets are chosen so as to maintain a conservative margin with respect to the relevant SIS- and **RR-LWR**-based attacks, rather than optimizing only for minimal sizes. This design philosophy is adopted for long-term evaluation and deployment.

Third, compared to ML-DSA, we simplify several low-level components that are highly relevant in practice. Since our secret coefficients are sampled from intervals $[-2, 1]$ or $[-4, 3]$, we sample secrets by generating 2 or 3 bits at a time without the need for rejection. Similarly, by choosing $q = 2^{24}$, public-matrix coefficients can be generated in a bit-exact way from pseudorandom output, again without discarding random bits. This is a small algorithmic change, but it has important practical consequences: it results in straightforward constant-time sampling routines, improves reproducibility across implementations, and reduces the amount of control-flow variability that implementers must harden. In contrast, even when the underlying distribution is simple, non-power-of-two ranges typically force additional rejection logic.

Fourth, we deliberately choose γ_1 and γ_2 to be powers of two. This replaces a number of decomposition-related operations by simple rounding operations and bit shifts. As a result, the analogue of ML-DSA’s **HighBits**, **LowBits**, and **Decompose** functionality can be expressed through a more regular bit-level interface. This is beneficial not only for software simplicity, but also for implementation assurance: arithmetic expressed through shifts, masks, and additions is generally easier to validate, optimize, and harden than arithmetic that depends on modular decomposition rules.

Fifth, we increase the ring degree to $n = 1024$. This has two distinct advantages. On the one hand, it increases the amount of ring structure available to the implementation, which improves the efficiency of the underlying arithmetic. On the other hand, it directly affects the challenge generation process: since **SampleInBall** places τ non-zero coefficients in a polynomial of degree n , a larger value of n allows us to reduce τ while maintaining the required challenge entropy. This in turn reduces $\beta = \tau\eta$, thereby enlarging the admissible interval for the response and increasing the success probability of the Fiat–Shamir abort loop. In other words, the larger ring dimension is used not merely for arithmetic efficiency, but also to improve signing efficiency and to reduce the variance of signing time. For example, at the 256-bit security level, the expected number of repetitions decreases using this design from roughly 3.85 to approximately 2.3. This

is a meaningful practical advantage: lower rejection rates reduce average signing cost, narrow timing variability, and simplify integration in systems with tighter latency constraints.

MLWRSign. A previous LWR-based signature scheme, MLWRSign, was proposed by Okada et al. [OTF⁺20]. The main difference is the treatment of the rounding error.

MLWRSign represents the rounding discrepancy through the rational quantity

$$\mathbf{s}_2^{\text{MLWRSign}} = \lfloor \frac{p}{q} \mathbf{A} \mathbf{s}_1 \rfloor - \frac{p}{q} \mathbf{A} \mathbf{s}_1 \in \mathbf{S}_{q/p, n, k},$$

and consequently introduces the auxiliary error terms

$$\boldsymbol{\xi}_1 = \lfloor \frac{p}{q} \mathbf{A} \mathbf{y} \rfloor - \frac{p}{q} \mathbf{A} \mathbf{y}, \quad \boldsymbol{\xi}_2 = \lfloor c \mathbf{s}_2 \rfloor - c \mathbf{s}_2, \quad \boldsymbol{\nu} = \lfloor \boldsymbol{\xi}_2 - \boldsymbol{\xi}_1 \rfloor.$$

to express the signing and verification equations. By contrast, we define

$$\mathbf{s}_2 = \mathbf{A} \mathbf{s}_1 - \frac{q}{p} \mathbf{b} \in \mathbf{S}_{q/p, n, k},$$

where $\mathbf{b} = \lfloor \frac{p}{q} \mathbf{A} \mathbf{s}_1 \rfloor$. The two definitions are related by $\mathbf{s}_2 = -\frac{q}{p} \mathbf{s}_2^{\text{MLWRSign}}$, so that $\mathbf{s}_2$ is an integer element of $\mathbf{S}_{q/p, n, k}$.

The two definitions contain the same information up to the known scaling $\mathbf{s}_2 = -\frac{q}{p} \mathbf{s}_2^{\text{MLWRSign}}$, but the algorithmic consequences are quite different. Our formulation avoids rational arithmetic entirely, eliminates the need for the extra bookkeeping variables $\boldsymbol{\xi}_1$, $\boldsymbol{\xi}_2$, and $\boldsymbol{\nu}$, and keeps the scheme much closer to the standard Fiat-Shamir with aborts template.

A second advantage is that our definition makes the distributional role of the rounding error more transparent. The coefficients of $\mathbf{s}_2$ are always bounded in the integer interval $[-q/(2p), q/(2p) - 1]$. Heuristically, if one treats $\mathbf{A} \mathbf{s}_1$ as uniform modulo q , this derived error is close to uniform on that interval; this stronger uniform-product heuristic is distinct from the RR-LWR assumption on the rounded value $\mathbf{b}$. Independently of this heuristic, the public key $(\mathbf{A}, \mathbf{b})$ is, by construction, an RR-LWR sample with deterministic error $\mathbf{s}_2$. This makes the connection between the algorithm and the underlying hardness assumption particularly explicit.

Finally, MLWRSign recomputes the rounding error during each signing invocation. We instead compute $\mathbf{s}_2$ once during key generation and store it in the secret key, exactly because it is fully determined by $\mathbf{A}$ and $\mathbf{s}_1$. This shifts a small amount of work and storage to key generation, but avoids repeated computation during signing. We consider this a favorable trade-off: key generation is performed rarely, whereas signing is the online operation in most deployment scenarios. Storing $\mathbf{s}_2$ therefore reduces per-signature cost, simplifies the signing code path, and makes the dynamic behavior of the implementation more regular. From an engineering point of view, this is again aligned with our broader design goal of prioritizing online efficiency and implementation robustness over minor savings in secret-key size.

Overall, Octarine can be viewed as combining the architectural clarity of ML-DSA with a cleaner LWR-based formulation than MLWRSign. Relative to ML-DSA, the main advantages are the use of a power-of-two modulus, simpler sampling and decomposition procedures, improved signing efficiency through a reduced abort rate, and substantially smaller matrix representations. Relative to MLWRSign, the main advantages are the elimination of rational arithmetic and auxiliary correction terms, a more direct interpretation of the public key as an RR-LWR sample, and lower signing cost by precomputing the rounding error during key generation. We believe these differences amount to a materially simpler and more implementation-oriented design, rather than a minor reformulation of existing techniques.

2.3 Security

In the classical random oracle model (ROM), we prove that the scheme is SUF-CMA secure and has the BUFF ownership properties together with the corresponding weak non-resignability

Scheme	Security				Size (B)		
	Attack	BKZ (b)	C	Q	sk	pk	sig
Octarine-128	key recovery	424	124	112	2 432	1 344	2 564
	unforgeability	439	128	116			
Octarine-256	key recovery	924	270	245	4 608	2 368	5 449
	unforgeability	910	266	241			
ML-DSA-44	key recovery	433	126	115	2 560	1 312	2 420
	unforgeability	423	124	112			
ML-DSA-87	key recovery	883	258	234	4 896	2 592	4 627
	unforgeability	909	265	241			
Falcon-512	key recovery	458	133	121	1 281	897	666
	unforgeability	411	120	108			
Falcon-1024	key recovery	936	273	248	2 305	1 793	1 280
	unforgeability	952	277	252			
HAETAE-120	key recovery	428	125	109	1 376	992	1 463
	unforgeability	409	119	105			
HAETAE-260	key recovery	988	288	253	2 720	2 080	2 908
	unforgeability	878	256	225			

Table 1: Security estimates for key recovery (LWR) and existential unforgeability (SIS) and size comparison in bytes between **Octarine** and related lattice-based schemes.

statement in the idealized high-min-entropy model. The unforgeability reduction uses the **RR-LWR** assumption and a SIS-type assumption, while the **BUFF**-style properties follow from the checked challenge digest under the stated hash-function and encoding assumptions. The **RR-LWR** problem is the radical-ring analogue of Module-LWR and is formally defined in Section 6.1.1. Section 6 gives the full security analysis, including the ROM security reductions and concrete classical and quantum attack-cost estimates for the proposed parameter sets.

We provide a high-level comparison with other lattice-based signature schemes in Table 1. The comparison reports classical and quantum Core-SVP costs for secret-key recovery and for the short-solution relations associated with signature forgery. The forgery column uses the EUF-CMA estimate. Corresponding SUF-CMA estimates may differ because strong unforgeability can require an additional homogeneous short-solution relation; this difference is particularly significant for HAETAE.

3 Preliminaries

This section presents all background material required for the specification.

3.1 Notation

Ring elements. The ring of integers is denoted $\mathbf{Z}$ and for a positive integer q , $\mathbf{Z}_q$ denotes the quotient ring $\mathbf{Z}/q\mathbf{Z}$. Given a positive integer m , the m -th cyclotomic polynomial, denoted by $\Phi_m(x)$, is defined as the unique monic polynomial whose roots are the primitive m -th roots of unity, in particular,

$$\Phi_m(x) = \prod_{\substack{k=1 \\ \gcd(k,m)=1}}^m \left(x - e^{2\pi i \frac{k}{m}} \right),$$

where $e^{2\pi i \frac{k}{m}}$ for $\gcd(k, m) = 1$ are the primitive m -th roots of unity. It can be shown that $\Phi_m(x)$ has integral coefficients and from the definition it follows that $\deg(\Phi_m(x)) = \varphi(m)$, where $\varphi(m)$ is Euler's totient function, which counts the number of positive integers less than m that are relatively prime to m . For $m = 2n$ with n itself a power of 2, it is well known that $\Phi_m(x) = x^n + 1$. The polynomial quotient ring is denoted as $\mathbf{R}_q = \mathbf{Z}_q[x]/(f(x))$ where $f(x) \in \mathbf{Z}[x]$ is monic.

Let n be a power of two and k a positive integer. We define the cyclotomic ring $\mathbf{R}_{q,n} = \mathbf{Z}_q[y]/(y^n + 1)$ and

$$\mathbf{S}_{q,n,k} = \mathbf{R}_{q,n}[x] / (x^k - y - 2)$$

as the degree- k radical extension of $\mathbf{R}_{q,n}$. Elements of $\mathbf{S}_{q,n,k}$ are represented as polynomials $\mathbf{g} = \sum_{i=0}^{k-1} \mathbf{g}_i x^i$ with coefficients $\mathbf{g}_i \in \mathbf{R}_{q,n}$. Regular lower-case letters denote elements of $\mathbf{Z}$ or $\mathbf{Z}_q$. Bold lower-case letters denote elements of $\mathbf{R}_{q,n}$ or $\mathbf{S}_{q,n,k}$ (polynomial or ring elements). Bold upper-case letters denote matrices.

Modular representations. For an even integer α and $r \in \mathbf{Z}$, we write $r' = r \bmod \alpha$ to denote the unique representative r' in the range $[-\alpha/2, \alpha/2)$ satisfying $r' \equiv r \pmod{\alpha}$, including $-\alpha/2$ and excluding $\alpha/2$. For α a power of two, this is equivalent to extracting the least-significant $\log_2(\alpha)$ bits of r . Unless otherwise mentioned, we assume all variables to be represented in a *signed* interval. If this operation is implemented in a modern CPU with 32- or 64-bit words, one has to be mindful of sign extension if top bits are dropped. We extend this operation coefficient-wise to elements of $\mathbf{S}_{q,n,k}$.

Norms. For a ring element $\mathbf{g} = \sum_{i=0}^{k-1} \mathbf{g}_i x^i \in \mathbf{S}_{q,n,k}$ and $\mathbf{g}_i = \sum_{j=0}^{n-1} \mathbf{g}_{i,j} y^j \in \mathbf{R}_{q,n}$, we define $\|\mathbf{g}\|_\infty = \max_{i,j} |\mathbf{g}_{i,j} \bmod q|$.

Sampling distributions. We use $D_{\mathbf{x}}(R)$ to denote the distribution used to sample the variable $\mathbf{x}$ from the ring R . If the sampling is done deterministically from a seed σ , we use the notation $D_{\mathbf{x}}(R; \sigma)$. If multiple seeds $\sigma_1, \dots, \sigma_m$ are used we write $D_{\mathbf{x}}(R; \sigma_1, \dots, \sigma_m)$ and they are concatenated into $\sigma = \sigma_1 \mid \dots \mid \sigma_m$ before sampling. For example, this is used to append ring indices to the various seeds.

Hash functions. The algorithms use hash functions $\mathbf{H}(\text{input}, \text{outlen})$ with variable (extendable) output length, where outlen is given in bytes. The output length of $\mathbf{H}$ is specified separately for each invocation. The security proofs require $\mathbf{H}$ to behave as a random oracle with $\lambda/2$ -bit collision resistance and λ -bit preimage resistance, where λ is the target security level. The concrete instantiation of $\mathbf{H}$ will be specified together with the final parameter sets.

3.2 Lattice-based Cryptography

Understanding the theoretical and practical aspects of LWR requires an introduction to the learning with errors (LWE) problem [Reg05] and its more popular structured variants: ring learning with errors [LPR10] and module learning with errors [BGV12, LS15]. We refer the interested reader to the comprehensive surveys related to LWE [Reg10, Pei16].

3.2.1 Learning with Errors

Let the module rank d be a positive integer and let χ denote a distribution over $\mathbf{R}$. Let $s \in \mathbf{R}_q^d$ be a secret vector and define the distribution $A_{s,\chi}$ as

$$A_{s,\chi} = (a, b = \langle a, s \rangle + e \bmod q) \in \mathbf{R}_q^d \times \mathbf{R}_q$$

where $a \leftarrow \mathcal{U}(\mathbf{R}_q^d)$ is sampled uniformly at random and $e \leftarrow \chi$ is the error term.

The *search LWE problem* is given access to polynomially many independent samples from $A_{s,\chi}$, recover the secret s , while the *decision LWE problem* is to distinguish between samples drawn from $A_{s,\chi}$ and samples drawn uniformly from $\mathbf{R}_q^d \times \mathbf{R}_q$. In practice, one often takes $\mathbf{R}_{q,n} = \mathbf{Z}_q[x]/(f(x))$ with $f(x)$ the cyclotomic polynomial

$$f(x) = \Phi_{2n}(x) = x^n + 1$$

with n a power of two. In this Ring-LWE variant all operations are performed within the ring $\mathbf{R}_q$. A single Ring-LWE sample embeds n Plain-LWE samples worth of information while drastically reducing the size of cryptographic keys. Furthermore, the choice of the cyclotomic polynomial allows for the polynomial multiplication to be computed very efficiently using the Number Theoretic Transform (NTT) [Pol71], a variant of the Fourier Transform that involves only integer arithmetic.

The Module-LWE problem [BGV12, LS15] generalizes both Plain-LWE and Ring-LWE, offering a flexible intermediate structure which enables a trade-off between security and efficiency. In Module-LWE, the secrets are vectors of polynomials and a Module-LWE sample consists of $(\mathbf{a}, b)$, where $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$. Hence, Module-LWE is a generalization of the Plain- and Ring-LWE versions. Setting the rank $d = 1$ specializes to the Ring-LWE problem whereas setting the ring to be $\mathbf{Z}$ and the rank $d = n$ particularizes to the Plain-LWE problem. Additionally, for a given parameter choice of a Module-LWE, the difficulty of the problem, i.e., the security of the scheme, can be adjusted by simply tuning the module rank.

3.2.2 Learning with Rounding

Introduced in [BPR12], learning with rounding can be viewed as a derandomization technique for LWE. In LWE one computes $\langle a, s \rangle + e \bmod q$: the error term e is *added* to the inner product (modulo q). Instead of this addition, the idea of LWR is to *scale and round* this inner product to some smaller group $\mathbf{Z}_p$. Specifically, this rounding divides the elements in $\mathbf{Z}_q$ up into p contiguous intervals and maps an element of $\mathbf{Z}_q$ to its corresponding index in $\mathbf{Z}_p$. The value of p needs to be smaller than q , otherwise, no error is introduced. This rounded inner product $\lfloor \frac{p}{q} \langle a, s \rangle \rfloor$ is denoted as

$$\lfloor \langle a, s \rangle \rfloor_p = \left\lfloor \frac{p}{q} \langle a, s \rangle \right\rfloor \in \mathbf{Z}_p.$$

Here $\mathbf{Z}_q$ (resp. $\mathbf{Z}_p$) is represented by $[0, q[$ (resp. $[0, p[$). Note that when $p \mid q$, the rounded version of the uniform distribution on $\mathbf{Z}_q$ is again uniform on $\mathbf{Z}_p$. The rounding operation can be extended to vectors / polynomials by applying it coefficient-wise. Analogously to the ring and module variants of learning with errors, one can define ring and module versions of LWR.

The decision version of the Learning with Rounding problem is as follows.

Definition 1 (Decision Learning with Rounding, based on [BPR12, Definition 3.1]). *Let $n \geq 1$ and moduli $q > p \geq 2$ be integers. Let $\mathbf{R}_q = \mathbf{Z}_q[x]/f(x)$ be as defined above.*

- **(Plain-LWR):** For a vector $s \in \mathbf{Z}_q^n$, define the LWR distribution L_s to be the distribution over $\mathbf{Z}_q^n \times \mathbf{Z}_p$ obtained by choosing a vector $a \leftarrow \mathbf{Z}_q^n$ uniformly at random, and outputting $(a, b = \lfloor \langle a, s \rangle \rfloor_p)$.
- **(Ring-LWR):** For a polynomial $s \in \mathbf{R}_q$, define the ring-LWR distribution L_s to be the distribution over $\mathbf{R}_q \times \mathbf{R}_p$ obtained by choosing $a \leftarrow \mathbf{R}_q$ uniformly at random and outputting $(a, b = \lfloor a \cdot s \rfloor_p)$.
- **(Module-LWR):** Fix a module rank $d > 1$. For $s \in \mathbf{R}_q^d$, define the module-LWR distribution L_s to be the distribution over $\mathbf{R}_q^d \times \mathbf{R}_p$ obtained by choosing $a \leftarrow \mathbf{R}_q^d$ uniformly at random and outputting $(a, \lfloor \langle a, s \rangle \rfloor_p)$.

The **decision (ring/module-) LWR** $_{n,q,p}$ problem is to distinguish (with non-negligible advantage in n) between polynomially many independent samples $(a_i, b_i) \leftarrow L_s$ and the same number of samples drawn uniformly and independently from $\mathbf{Z}_q^n \times \mathbf{Z}_p$, $\mathbf{R}_q \times \mathbf{R}_p$, or $\mathbf{R}_q^d \times \mathbf{R}_p$, respectively.

3.2.3 Security of Learning with Rounding

The high-level idea for the security of LWR is to show that the security is statistically close to that of solving decision-LWE (distinguishing from random $\mathbf{u}$) [AKPW13]:

$$(a, \lfloor \langle a, s \rangle \rfloor_p) \stackrel{\text{stat}}{\approx} (a, \langle a, s \rangle + e) \stackrel{\text{comp}}{\approx} (a, \mathbf{u}).$$

The security of LWR was initially studied in [BPR12]. This paper shows the following result.

Theorem 1 (Reduction to LWE [BPR12, Theorem 3.2]). *Let χ_n be any efficiently sampleable family of probability distributions, indexed by the security parameter n , over $\mathbf{Z}$ which is $\beta(n)$ -bounded (that is, $\Pr_{x \leftarrow \chi} [|x| > \beta] \leq \text{negl}(n)$). Let $q \geq p \cdot \beta \cdot n^{\omega(1)}$. Then for any distribution over the secrets $s \in \mathbf{Z}_q^n$ solving decision-LWR $_{n,q,p}$ is at least as hard as solving decision-LWE $_{n,q,\chi}$ for the same distribution over s .*

Theorem 1 requires that the modulus $q \geq p \cdot \beta \cdot n^{\omega(1)}$ is super-polynomial. This result was improved by [AKPW13] by providing a security reduction that works for a larger range of parameters: the size of the modulus q used is not super-polynomial but polynomial [AKPW13, Thm. 4.1+Cor. 4.2]. Specifically, the authors show the hardness of LWR using parameters n , m (the number of samples), q , and p assuming the hardness of LWE parameters n' , m , q , β as long as

$$n \geq \frac{\log(q)}{\log(2\gamma)} \cdot n' \quad \text{and} \quad q \geq \gamma(nm\beta p)$$

for some parameter $\gamma \geq 1$. For the reduction to work the modulus q needs to be much larger than the number of LWR samples m . Hence, when an attacker is allowed to choose the number of LWR samples (and is not fixed a priori) the modulus q becomes super-polynomial again. The modulus q needs to be prime or the above inequality should hold for the largest prime divisor $q_{\max}$ of q . Moreover, the result requires q and $q/q_{\max}$ to be co-prime: this implies that the result does not hold when the modulus q is a power of two.

This restriction is lifted by the results from [BGM⁺16]. Firstly, the bound on the modulus q is improved to

$$q \geq 2m\beta p$$

and secondly, this result holds without any number-theoretic restrictions on q : specifically, the result holds when the modulus q is a power of two [BGM⁺16, Thm. 1 + Thm. 3].

Theorem 2 (One-wayness of LWR [BGM⁺16, Theorem 1]). *Let p , q , n , m , and β be integers such that $q > 2p\beta$. For every algorithm Learn ,*

$$\Pr_{A,s,e} [\text{Learn}(A, \lfloor As + e \rfloor_p) = s] \geq \frac{\Pr_{A,s} [\text{Learn}(A, \lfloor As \rfloor_p) = s]^2}{(1 + 2p\beta/q)^m},$$

where $A \leftarrow \mathbf{Z}_q^{m \times n}$, the noise e is independent over all m coordinates, β -bounded and balanced in each coordinate, and s is chosen from any distribution supported on $\mathbf{Z}_q^{n*}$.

This result proves the hardness of LWR when the number of samples m is bounded. More recently, the authors of [NR23] studied the unbounded setting and in this regime identify a barrier using “pointwise” reductions from LWE to LWR. The main result of this work and the associated conjecture require the modulus q to be prime [NR23, Thm. 1 + Conj. 1]: which is *not* the case in our design. It should be noted that the results of [NR23] indicate new ideas are necessary to prove the security in the unbounded sample setting and *does not* suggest that LWR is easy.

3.3 Rings as Structured Modules

In the setting of Module-LWR a single sample is of the form $[\langle \mathbf{A}, \mathbf{s} \rangle]_p$, where $\mathbf{A}, \mathbf{s} \in \mathbf{R}_{q,n}^k$, with k the rank of the module. The samples are selected such that $\mathbf{A}$ is uniformly distributed and $\mathbf{s}$ and $\mathbf{e}$ are generated from a “small” distribution. If m samples are selected then the security level is equivalent to a Plain-LWR problem with matrix of size $mn \times kn$, under the assumption that the ring and module structure cannot be exploited. By selecting $m = k$ one recovers the setting of typical KEMs, where the learning with rounding problem is defined by a square matrix. For simplicity we assume that $m = k$ from here onwards.

The main advantage of Module-LWR is that k can be arbitrarily chosen. For example, Saber selects $k \in \{2, 3, 4\}$. For contrast, Ring-LWR based on a cyclotomic ring achieving the same security level would sample from $R_{q,kn}$ and therefore must restrict k to powers of two and cannot select $k = 3$. On the other hand, the advantage of Ring-LWR is that a sample element $\mathbf{A}$ contains only kn different elements in $\mathbf{Z}_q$ (a single element $\mathbf{A} \in R_{q,kn}$) compared to k^2n for Module-LWR (k samples of elements in $\mathbf{R}_{q,n}^k$). The quadratic scaling of Module-LWR reduces the efficiency compared to Ring-LWR, but the greater flexibility with k allows for easier instantiations of concrete parameter sets.

It is well known how Ring-LWR corresponds to (or can be viewed as) a structured Module-LWR instance. For each power of two $k \mid n$ we have the isomorphism

$$\mathbf{R}_{q,n} = \mathbf{Z}_q[x]/(x^n + 1) \cong \left(\mathbf{Z}_q[y]/(y^{\frac{n}{k}} + 1) \right) [x]/(x^k - y) \quad (2)$$

by sending

$$s(x) = \sum_{i=0}^n s_i x^i \mapsto \sum_{j=0}^{k-1} s_j(y) \cdot x^j, \quad \text{where} \quad s_j(y) = \sum_{i=0}^{n/k-1} s_{ki+j} \cdot y^i.$$

This simply divides the polynomial s up into k parts $s_j(y)$, where each $s_j(y)$ contains the coefficients with degree equal to j modulo k . For example, for $k = 2$ we have $s_0(y)$ containing the coefficients of *even* degree and $s_1(y)$ the coefficients of *odd* degree. Taking two elements $\mathbf{A}, \mathbf{s} \in \mathbf{R}_{q,n}$, under this isomorphism we see that

$$\mathbf{A} \cdot \mathbf{s} = \begin{bmatrix} a_0(y) & ya_1(y) \\ a_1(y) & a_0(y) \end{bmatrix} \begin{bmatrix} s_0(y) \\ s_1(y) \end{bmatrix},$$

where each operation takes place in the ring $\mathbf{Z}_q[y]/(y^{\frac{n}{2}} + 1)$, and in particular ya_1 corresponds to an anti-cyclic rotation of the coefficients of $a_1(y)$. This follows from the fact that

$$\begin{aligned} \mathbf{A} \cdot \mathbf{s} &= (a_0(y) + a_1(y)x) \cdot (s_0(y) + s_1(y)x) \\ &= a_0s_0(y) + (a_0s_1(y) + a_1s_0(y))x + a_1s_1(y)x^2 \\ &= a_0s_0(y) + ya_1s_1(y) + (a_0s_1(y) + a_1s_0(y))x \pmod{x^2 - y}. \end{aligned}$$

For $k = 4$ we would instead get

$$\mathbf{A} \cdot \mathbf{s} = \begin{bmatrix} a_0(y) & y \cdot a_3(y) & y \cdot a_2(y) & y \cdot a_1(y) \\ a_1(y) & a_0(y) & y \cdot a_3(y) & y \cdot a_2(y) \\ a_2(y) & a_1(y) & a_0(y) & y \cdot a_3(y) \\ a_3(y) & a_2(y) & a_1(y) & a_0(y) \end{bmatrix} \begin{bmatrix} s_0(y) \\ s_1(y) \\ s_2(y) \\ s_3(y) \end{bmatrix}, \quad (3)$$

operations now taking place in $\mathbf{Z}_q[y]/(y^{\frac{n}{4}} + 1)$. More generally, it is easy to see that $\mathbf{A} \cdot \mathbf{s}$ corresponds to a multiplication of a $k \times k$ circulant matrix with upper right triangle multiplied by y . For comparison, for $k = 4$, a Module-LWR matrix multiplication is structured as

$$\begin{bmatrix} a_{0,0}(y) & a_{0,1}(y) & a_{0,2}(y) & a_{0,3}(y) \\ a_{1,0}(y) & a_{1,1}(y) & a_{1,2}(y) & a_{1,3}(y) \\ a_{2,0}(y) & a_{2,1}(y) & a_{2,2}(y) & a_{2,3}(y) \\ a_{3,0}(y) & a_{3,1}(y) & a_{3,2}(y) & a_{3,3}(y) \end{bmatrix} \begin{bmatrix} s_0(y) \\ s_1(y) \\ s_2(y) \\ s_3(y) \end{bmatrix},$$

where arithmetic also takes place in the ring $\mathbf{Z}_q[y]/(y^{\frac{n}{4}} + 1)$.

To summarize, for any $k \mid n$ one can view a cyclotomic Ring-LWR sample as a structured Module-LWR sample of rank k where

$$a_{i,j}(y) = \begin{cases} \text{uniformly random,} & \text{for } j = 0, \\ y \cdot a_{k+i-j,0}(y), & \text{for } 0 \leq i < j < k, \\ a_{i-j,0}(y), & \text{for } 0 < j \leq i < k. \end{cases} \quad (4)$$

The above reinterpretation is purely syntactic and has no impact on the security. Under the ring isomorphism in Eq. (2), a cyclotomic Ring-LWR sample is mapped bijectively to the structured Module-LWR instance of rank k described in Eq. (4). Any distinguisher or secret-recovery algorithm for structured Module-LWR immediately yields one for Ring-LWR by composing with the inverse isomorphism, with identical advantage and essentially the same running time. Hence, viewing a cyclotomic Ring-LWR sample as a structured Module-LWR sample of rank k does not alter the underlying hardness assumptions nor concrete estimates: it is simply a different representation of the same distribution (and the same reasoning applies to LWE).

3.4 Radical Ring-LWR

The observations from Section 3.3 form the key ingredient for defining a different way to interpret and define new rings and parameters for Ring-LWR and Module-LWR based schemes.

Instead of using a power-of-two cyclotomic $f(x) = x^n + 1$, we select the polynomial $f(x) = (x^k - 2)^n + 1$ for a power of two n and strictly positive integer k . It can be shown that f is irreducible [BRVW26, Lemma 1], implying that $\mathbf{Q}[x]/(f(x))$ is a number field and as such we can define a Ring-LWR-type problem for this $f(x)$. In the ARCANE instantiation, rounding is applied coefficient-wise in the transported monomial basis, namely the image of the radical-ring monomial basis $\{y^i x^j\}$ under the number-field isomorphism; the connection with the usual canonical-embedding viewpoint is handled in the security section by bounding the corresponding basis distortion. A priori it may seem that the change of polynomial removes the advantages that power-of-two cyclotomics offer. However, there exists a simple ring isomorphism

$$S_{q,n,k} := \mathbf{Z}_q[x]/((x^k - 2)^n + 1) \cong (\mathbf{Z}_q[y]/(y^n + 1)[x]) / (x^k - y - 2), \quad (5)$$

$$x^k \mapsto y + 2.$$

While $S_{q,n,k}$ is not a cyclotomic ring itself, it has a large power-of-two cyclotomic subring in which arithmetic optimizations (such as NTTs) can still be used. The field defined by $f(x)$ is thus a degree k extension of the $2n$ -th cyclotomic field and the role of the extension degree k is similar to that of a module rank, allowing easier scaling of the hardness of the underlying lattice problem than for cyclotomic rings in which n must be a power of two. Since the polynomial $x^k - y - 2$ defines a degree k radical extension of the $2n$ -th cyclotomic field this type of ring is referred to as *radical* Ring-LWR (RR-LWR) in the learning with rounding setting.

Note that even if one would consider a more general m -th cyclotomic field instead of just a power of two n , it would still be impossible to construct cyclotomic fields with a large power-of-two cyclotomic subfield and of given degree. Indeed, since the degree of the m -th cyclotomic field is given by Euler $\varphi(m)$, the possible extension degrees are limited to integers of this form and even more so when a large power-of-two subfield is required for efficiency reasons. To illustrate this: since there are no integers m with $\varphi(m) = 3$ or $\varphi(m) = 5$, there do not exist cyclotomic fields that correspond to degree 3 or 5 extensions of a two-power cyclotomic field. With the above choice of $f(x)$ such fields are immediate simply by setting $k = 3$ or $k = 5$.

Compared to using a generic module structure, the above construction has the advantage of imposing a ring structure that reduces the computational complexity and randomness compared to Module-LWR. That is, the matrix $\mathbf{A}$ consists of k as opposed to k^2 uniformly random elements in the cyclotomic ring. More concretely, for elements $\mathbf{A}, \mathbf{s} \in S_{q,n,k}$ and writing $\mathbf{A} = \sum_{i=0}^{k-1} a_i(y)x^i$

and $\mathbf{s} = \sum_{i=0}^{k-1} s_i(y)x^i$, the polynomial multiplication $\mathbf{A} \cdot \mathbf{s}$ corresponds to the matrix multiplication

$$\begin{bmatrix} a_0(y) & (y+2) \cdot a_{k-1}(y) & (y+2) \cdot a_{k-2}(y) & \cdots & (y+2) \cdot a_1(y) \\ a_1(y) & a_0(y) & (y+2) \cdot a_{k-1}(y) & \cdots & (y+2) \cdot a_2(y) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k-1}(y) & a_{k-2}(y) & a_{k-3}(y) & \cdots & a_0(y) \end{bmatrix} \begin{bmatrix} s_0(y) \\ s_1(y) \\ s_2(y) \\ \vdots \\ s_{k-1}(y) \end{bmatrix}. \quad (6)$$

All operations in this matrix multiplication take place in the cyclotomic ring $\mathbf{Z}_q[y]/(y^n + 1)$. Compared to the Ring-LWR matrix structure in Equation (3), we see that it essentially simply replaces the multiplication factor y by $y + 2$.

One has to be careful how to instantiate Ring-LWR over a number ring different from a cyclotomic ring; in the security section we formalize our instantiation as a transported-basis Ring-LWR variant over the radical number field and account for the basis change used in the concrete estimates.

4 Algorithm Description

In this section we describe all the components that are needed to build the **Octarine** digital signature scheme. We specify the full algorithms in Algorithm 3.

Parameter sets and security levels. The algorithms are parameterized to support multiple security levels in accordance with the submission requirements. In particular, three parameter sets are specified to target classical security strengths of 128, 256, and 512 bits, respectively. For each parameter set, the corresponding quantum-resistant security strength is estimated to be no less than 80, 128, and 256 bits.

Each parameter set instantiates the same algorithmic structure, differing only in the choice of parameters that determine the underlying lattice dimension and modulus sizes, and some of the constants related to the signing algorithm. This unified design enables a consistent implementation framework across all security levels while allowing fine-grained scaling of security through parameter selection. The full list of parameters is summarized in Table 2.

Ring Selection. We operate over a radical ring of the form

$$\mathbf{S}_{q,n,k} = (\mathbf{Z}_q[y]/(y^n + 1))[x]/(x^k - y - 2), \quad (7)$$

where q and n are powers of two, and k is an arbitrary positive integer. This is the same ring as used by **Mithril**. There exists a straightforward isomorphism to the number-field representation $\mathbf{S}_{q,n,k} \cong \mathbf{Z}_q[x]/((x^k - 2)^n + 1)$ [BRVW26, Section 4], giving the radical-ring representation used for our transported-basis Ring-LWR variant. The ring $\mathbf{S}_{q,n,k}$ is a degree k *radical* extension of the $2n$ -th cyclotomic field. As such, standard implementation techniques (such as NTTs or other polynomial multiplication routines) can be used if desired, while the radical extension allows more efficient matrix generation than a module structure.

We unify the choice of $q = 2^{24}$ and $n = 1024$ across the parameter sets and vary k to obtain the appropriate matrix dimensions. We select q to have a bit length that is a multiple of 8 to simplify implementation, and maximize n to allow for the use of NTTs if preferred. Moreover, as q is exactly double the size of the prime used by **Mithril**, we can re-use its building blocks more easily. Again, while this is an NTT-friendly ring, we design our schemes such that the use of NTT is completely optional. The Ring Learning-With-Rounding problem is defined by rounding error q/p . We select p such that $\mathbf{s}_1$ is sampled uniformly from $[-\eta, \eta - 1]$ and the derived rounding error $\mathbf{s}_2 = \mathbf{A}\mathbf{s}_1 - \frac{q}{p}\mathbf{b}$ is bounded in the same coefficient interval, where $\eta = q/(2p)$. Heuristically,

	Octarine-128	Octarine-256	Octarine-512
n	1024	1024	1024
k	1	2	5
$\epsilon_q = \log_2 q$	24	24	24
$\epsilon_p = \log_2 p$	21	22	22
η	4	2	2
d	11	13	14
τ	16	36	87
λ	128	256	512
γ_1	2^{18}	2^{19}	2^{21}
γ_2	2^{17}	2^{18}	2^{20}
$\beta = \tau\eta$	64	72	174
ω	79	210	399
Challenge entropy	131	257	512
# Repetitions	2.1	2.3	3.6
$D_{\mathbf{A}}$	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_{\epsilon_q}$
$D_{\mathbf{s}_1}$	$\mathcal{U}_4$	$\mathcal{U}_2$	$\mathcal{U}_2$
Private key (bytes)	2 432	4 608	11 776
Public key (bytes)	1 344	2 368	5 184
Signatures (bytes)	2 564	5 449	14 713
<u>LWR Hardness (uSVP)</u>			
BKZ block size (b)	434	935	2788
Classical Core-SVP ($0.292b$)	127	273	814
Quantum Core-SVP ($0.265b$)	115	248	739
Refined estimate (gates)	153	294	817
<u>LWR Hardness (BDD)</u>			
BKZ block size (b)	424	924	2181
Classical Core-SVP ($0.292b$)	124	270	637
Quantum Core-SVP ($0.265b$)	112	245	578
Refined estimate (gates)	150	291	686
<u>SIS Hardness (EUF-CMA; SUF-CMA in parentheses)</u>			
BKZ block size (b)	439 (419)	910 (916)	2340 (2340)
Classical Core-SVP ($0.292b$)	128 (122)	266 (267)	683 (683)
Quantum Core-SVP ($0.265b$)	116 (111)	241 (242)	620 (620)
Refined estimate (gates)	156 (150)	288 (290)	691 (692)

Table 2: Parameter sets, encoded sizes, and hardness estimates. For the SIS hardness rows, the main entries are the EUF-CMA estimates used in Table 1; the parenthesized entries are the corresponding SUF-CMA estimates.

if one treats $\mathbf{As}_1$ as uniform modulo q , this derived rounding error is close to uniform on that interval; this is a stronger uniform-product heuristic, distinct from the RR-LWR assumption on the rounded value $\mathbf{b}$. Each coefficient of the $\mathbf{s}_1$ and $\mathbf{s}_2$ can be represented with exactly $\log_2(2\eta)$ bits, which is either 2 or 3 bits for our parameter sets.

Hashing. Octarine requires hash functions, or more specifically eXtendable Output Functions (XOFs), for instantiating random oracles, message digest creation, as well as generating pseudo-random bits for key material and ephemeral secrets. All hash and XOF inputs are domain-separated and length-delimited.

First, let $P = \text{“ARCANE-Octarine-v1”}$ be a fixed protocol prefix. For a label L , output byte length o , and t byte-string fields $X_1, \dots, X_t$, the header $\text{Head}(L, o, t)$ is the concatenation of the fields below, where all integers are encoded little-endian.

Field	Value	Bytes
Protocol prefix	P	18
Label length	$ L $	1
Label	L	$ L $
Security level	λ	2
Module rank	k	1
Output length	o	4
Field count	t	1

Each variable-length field is prefixed with its own length, and a labelled input is the header followed by every encoded field:

$$\begin{aligned} \text{Field}(X) &= \text{IntegerToBytes}(|X|, 8) \parallel X, \\ \text{Encode}(L, o; X_1, \dots, X_t) &= \text{Head}(L, o, t) \parallel \text{Field}(X_1) \parallel \dots \parallel \text{Field}(X_t). \end{aligned}$$

We write

$$H_L(X_1, \dots, X_t; o) = \mathsf{H}(\text{Encode}(L, o; X_1, \dots, X_t), o).$$

The random-oracle calls used by the signature scheme are:

- *Key derivation:* $(\rho, \rho', K) \leftarrow H_{\text{kdf}}(\xi; |\rho| + |\rho'| + |K|)$ expands initial randomness into the public seed, secret seed, and signing key. The parameter-set values λ and k are included in the common encoding header.
- *Public key hash:* $tr \leftarrow H_{\text{tr}}(pk; |tr|)$ compresses the public key into a short digest for use in signing and verification.
- *Message representative:* $\mu \leftarrow H_{\text{mu}}(tr, M; |\mu|)$ pre-hashes the message together with tr , so that the signing algorithm processes a fixed-length input. In particular, the variable-length message M is encoded with its own length.
- *Signing randomness:* $\rho'' \leftarrow H_{\text{rhop}}(K, rnd, \mu; |\rho''|)$ derives per-attempt randomness for sampling $\mathbf{y}$. Including K hedges against poor randomness, including rnd provides nonce freshness.
- *Challenge hash:* $\tilde{c} \leftarrow H_{\text{ch}}(\mu, \mathbf{w}_1; |\tilde{c}|)$ derives the Fiat–Shamir challenge seed from the message representative and the commitment high bits.

The output length of H is specified separately for each invocation. The security proof requires H to behave as a random oracle with $\lambda/2$ -bit collision resistance and λ -bit preimage resistance, where λ is the target security level. We note that the security of the scheme does not depend on the specific choice of hash function, as long as the security requirements are satisfied. For the reference and optimized implementation families, we instantiate H with the SM3-based `pseudoXOF`. The additional SHA3/SHAKE implementation family instantiates H with `SHAKE256`. As `SHAKE256` only achieves 256-bit security at most due to its 512-bit capacity, the canonical 512-bit implementation uses the SM3-based `pseudoXOF` rather than `SHAKE256`.

Second, we also require a XOF for pseudo-random bit generation to expand private and public key material from seeds. This is used in the generation of $\mathbf{s}_1$, $\mathbf{A}$, $\mathbf{y}$, and c (for `SampleInBall`). We use XOF_A for the XOF on public data $\mathbf{A}$, XOF_S for secret sampling, and XOF_C for challenge expansion. The challenge-expansion XOF uses the same Encode format with label `XOF_C`. The high-volume seed-expansion XOFs XOF_A and XOF_S use a compact length-delimited encoding to avoid unnecessary extra compression blocks in the implementation:

$$\begin{aligned} \text{AOX1} \parallel \text{id} \parallel \text{IntegerToBytes}(\lambda, 2) \parallel \text{IntegerToBytes}(k, 1) \parallel \text{IntegerToBytes}(o, 2) \\ \parallel \text{IntegerToBytes}(|s|, 1) \parallel s \parallel \text{IntegerToBytes}(i, 1) \parallel \text{IntegerToBytes}(\ell, 1). \end{aligned}$$

Here $\text{id} = 1$ for `XOF_A`, $\text{id} = 2$ for `XOF_S`, s is the seed passed to the XOF call, i is the polynomial or stream index, and ℓ is the lane index. The seed lengths used by the scheme are below 256 bytes and the requested XOF outputs are below 2^{16} bytes, so this remains injective and length-delimited for every implemented call. The concrete XOF primitive underneath XOF_A is denoted G , and the primitive underneath XOF_S and XOF_C is denoted F . For the reference and optimized implementation families, we instantiate both G and F with the SM3-based `pseudoXOF`. In the additional SHA3/SHAKE implementation family we use `SHAKE128` for G and `SHAKE256` for F .

Sampling distributions. We sample secret and public values in various uniform distributions. Unless otherwise explicitly stated, we use *little* Endian for bit and byte addressing. As sampling a single polynomial with $n = 1024$ can require multiple sequential iterations of a permutation, this has the potential of creating a bottleneck. This is specifically the case for $\mathbf{A}$ and $\mathbf{y}$ where a relatively large amount of random bytes are needed. To allow for more parallel processing for $\mathbf{A}$ and $\mathbf{y}$, we generate each block of 256 polynomial coefficients independently using domain separation. We do not use this for $\mathbf{s}$ as only 2 or 3 bits per coefficient are required, so there is only little impact on performance.

- $D_A(\mathbf{S}_{q,n,k}, \rho)$: Deterministically expands the seed ρ to a uniformly random element of $\mathbf{S}_{q,n,k}$ using an extendable output function. In total, the generation of $\mathbf{A}$ requires k polynomials with n coefficients, each coefficient consisting of $\epsilon_q = 24$ bits. We write $\mathbf{A} = \sum_{i=0}^{k-1} \mathbf{A}_i(y)$ and $\mathbf{A}_i(y) = \sum_{j=0}^{n-1} \alpha_{i,j} y^j$. The sampling is done in two steps. First, we generate the byte stream from ρ . For each polynomial we use the index i and the four lane indices as separate encoded fields. That is, the byte stream for $\mathbf{A}_i(y)$ is generated as $B_{i,0} = XOF_A(\rho, \text{IntegerToBytes}(i, 1), \text{IntegerToBytes}(0, 1); 768), \dots, B_{i,3} = XOF_A(\rho, \text{IntegerToBytes}(i, 1), \text{IntegerToBytes}(3, 1); 768)$, each of the $B_{i,j}$ outputting $24 \cdot 256/8 = 768$ bytes. Second, we set $\alpha_{i,j} = B_{i,j'}[3j''] + B_{i,j'}[3j'' + 1] \cdot 2^8 + B_{i,j'}[3j'' + 2] \cdot 2^{16}$, where $j' = \lfloor j/256 \rfloor$ and $j'' = j \bmod 256$.
- $D_y(\mathbf{S}_{q,n,k}, \rho'', \kappa)$: Samples a ring element whose coefficients are drawn independently and uniformly from $[-\gamma_1, \gamma_1 - 1]$. In total, the generation of $\mathbf{y}$ requires k polynomials with n coefficients, each coefficient consisting of $\log_2(2\gamma_1)$ bits. The counter κ ensures that each signing attempt uses a fresh, independent sample. We write $\mathbf{y} = \sum_{i=0}^{k-1} \mathbf{y}_i(y)$ and $\mathbf{y}_i(y) = \sum_{j=0}^{n-1} \xi_{i,j} y^j$. The sampling is done in two steps. First, form the attempt seed $\rho''_\kappa = \rho'' \parallel \text{IntegerToBytes}(\kappa, 4)$. For each polynomial we use the index i and the four

lane indices as separate encoded fields. That is, the byte stream for $\mathbf{y}_i(y)$ is generated as $B_{i,0} = \text{XOF}_5(\rho''_\kappa, \text{IntegerToBytes}(i, 1), \text{IntegerToBytes}(0, 1); \log_2(2\gamma_1) \cdot 256/8), \dots, B_{i,3} = \text{XOF}_5(\rho''_\kappa, \text{IntegerToBytes}(i, 1), \text{IntegerToBytes}(3, 1); \log_2(2\gamma_1) \cdot 256/8)$, each of the $B_{i,j}$ outputting $\log_2(2\gamma_1) \cdot 256$ bits. Second, we set $\xi_{i,j} = B_{i,j'}[\log_2(2\gamma_1)j'' : \log_2(2\gamma_1)(j'' + 1)]$, where $B_{i,j'}[a : b]$ denotes bit index a to b , including a and excluding b , $j' = \lfloor j/256 \rfloor$ and $j'' = j \bmod 256$.

- $D_{\mathbf{s}}(\mathbf{S}_{q,n,k}, \rho')$: Samples a ring element whose nk integer coefficients are drawn independently and uniformly from $[-\eta, \eta - 1]$. In total, the generation of $\mathbf{s}$ requires k polynomials with n coefficients, each coefficient consisting of $\theta = \log_2(2\eta)$ bits. We write $\mathbf{s} = \sum_{i=0}^{k-1} \mathbf{s}_i(y)$ and $\mathbf{s}_i(y) = \sum_{j=0}^{n-1} \sigma_{i,j} y^j$. First, the seed ρ' is used by XOF_5 to output the k byte streams $B_i = \text{XOF}_5(\rho', \text{IntegerToBytes}(i, 1), \text{IntegerToBytes}(0, 1); \theta n/8)$. Second, we set $\sigma_{i,j} = B_i[\theta j : \theta(j + 1)]$, where $B_i[a : b]$ denotes bit index a to b , including a and excluding b .

Rounding and dividing. As usual, integers modulo a power-of-two coefficient modulus Q are represented in the interval $[-Q/2, Q/2 - 1]$. For any power-of-two divisor $m \mid Q$ and any $x \in [-Q/2, Q/2 - 1]$, we can uniquely write $x = x_0 + x_1 m \bmod Q$, where $x_0 \in [-m/2, m/2 - 1]$ and $x_1 \in [-Q/(2m), Q/(2m) - 1]$. This can be computed as $x_0 = x \bmod m$ and $x_1 = (x - x_0)/m \bmod Q/m$. We write the ambient modulus as a subscript on the rounding routines. Having this in mind, we define

- $\text{HighBits}_q(\mathbf{w}, 2\gamma_2)$ returns the ring element $\mathbf{w}_1 = ((\mathbf{w} - (\mathbf{w} \bmod 2\gamma_2)) / (2\gamma_2)) \bmod q / (2\gamma_2)$, where each operation is applied coefficient-wise.
- $\text{Power2Round}_p(\mathbf{b}, 2^d)$ returns the ring element $\mathbf{b}_0 = \text{LowBits}_p(\mathbf{b}, 2^d)$, where each operation is applied coefficient-wise, as well as $\mathbf{b}_1 = \text{HighBits}_p(\mathbf{b}, 2^d)$. The exponent d , and hence the modulus 2^d , is defined as part of the parameter set.
- $\mathbf{b} = \lfloor \frac{p}{q} \mathbf{t} \rfloor$ returns the rounded ring element. For an implementation-level description, let $\langle x \rangle_Q \in \{0, \dots, Q - 1\}$ denote the unsigned representative of $x \bmod Q$, and let $\text{ctr}_Q(u) = ((u + Q/2) \bmod Q) - Q/2$ denote the centered representative. Coefficient-wise, compute the rounded residue

$$u = (\langle \mathbf{t} \rangle_q + q/(2p)) \bmod q, \quad v = u \gg (\log_2 q - \log_2 p) \in \{0, \dots, p - 1\}, \quad \mathbf{b} = \text{ctr}_p(v).$$

Here $\gg$ denotes an unsigned right shift; an implementation may keep v as the unsigned representative of $\mathbf{b} \bmod p$. Equivalently, an implementation targeting an environment that guarantees arithmetic (sign-propagating) right-shift semantics for centered signed representatives may add $q/(2p)$, shift, and then reduce the result modulo p . Portable C implementations should instead form the unsigned representative before shifting, since right-shifting a negative signed integer is implementation-defined in C.

We refer to Algorithm 1 for algorithmic descriptions.

Public key compression. The public key stores only the high-order bits $\mathbf{b}_1$ of the LWR public key $\mathbf{b}$, obtained via Power2Round_p . During verification, the verifier must recover the high-order bits $\mathbf{w}_1 = \text{HighBits}_q(\mathbf{w}, 2\gamma_2)$ of the commitment $\mathbf{w} = \mathbf{A}\mathbf{y}$. However, since the verifier does not know $\mathbf{b}_0$, it can only compute the approximate value $\mathbf{w}'_{\text{Approx}} = \mathbf{w} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0$, which differs from $\mathbf{w} + c\mathbf{s}_2$ by the small quantity $c_p^q \mathbf{b}_0$. If this perturbation causes a carry in the high-order bits, the verifier would obtain the wrong value of $\mathbf{w}_1$. To correct for this, the signer includes a compact hint vector $\mathbf{h}$ that encodes the coefficient positions where carries occur, enabling the verifier to recover $\mathbf{w}_1$ exactly. That is, the hint encodes exactly the coefficient locations where

Algorithm 1 Supporting algorithms for the signature scheme.

<u>HighBits_Q(r, m)</u> 1: $r_0 \leftarrow r \bmod m$ 2: return $((r - r_0) / m) \bmod Q/m$ <u>LowBits_Q(r, m)</u> 1: return $r \bmod m$ <u>Power2Round_Q(r, m)</u> 1: $r_0 \leftarrow \text{LowBits}_Q(r, m)$ 2: $r_1 \leftarrow \text{HighBits}_Q(r, m)$ 3: return (r_1, r_0)	<u>MakeHint_Q(z, r, m)</u> 1: $r_1 \leftarrow \text{HighBits}_Q(r, m)$ 2: $v_1 \leftarrow \text{HighBits}_Q(r + z, m)$ 3: return $\llbracket r_1 \neq v_1 \rrbracket$ <u>UseHint_Q(h, r, m)</u> 1: $r_1, r_0 \leftarrow \text{Power2Round}_Q(r, m)$ 2: if $h = 1$ and $r_0 > 0$ then 3: return $(r_1 + 1) \bmod Q/m$ 4: else if $h = 1$ and $r_0 \leq 0$ then 5: return $(r_1 - 1) \bmod Q/m$ 6: end if 7: return r_1
---	---

* The subscript Q denotes the ambient coefficient modulus. In this scheme, Power2Round_p calls LowBits_p and HighBits_p with $m = 2^d$ on $\mathbf{b} \in S_p$; the remaining routines use $Q = q$ and $m = 2\gamma_2$. All operations are extended coefficient-wise to radical-ring elements and component-wise to vectors.

$\text{HighBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2)$ and $\text{HighBits}_q(\mathbf{w} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0, 2\gamma_2)$ differ. To account for this, the verifier increments $\mathbf{w}'_{\text{Approx}}$ if the hint coefficient is set, adding 1 if its lower-order bits are positive, and subtracting 1 if they are negative. We refer to Algorithm 1 for algorithmic descriptions of MakeHint_q and UseHint_q , which are extended coefficient-wise to full ring elements.

Since q and $2\gamma_2$ are both powers of two with $2\gamma_2 \mid q$, the decomposition into high-order and low-order parts is a clean bit extraction. Unlike the prime-modulus case (as in Dilithium [DKL⁺18] which later became the ML-DSA standard), there is no boundary anomaly at $q - 1$ and no need for the special-purpose Decompose subroutine, consequently, the hint mechanism simplifies to a single carry-direction check.

As d increases we compress the public key more and more, increasing the failure rate of the signing procedure. We select d heuristically, taking the largest value for which implementation measurements show only rare rejections from the $c_p^q \mathbf{b}_0$ -bound test and the subsequent hint-weight test. This choice has no impact on security and can be repeated with the reference implementation.

Key and signature encoding. We serialize keys and signatures as packed byte arrays in little Endian. Recall that we represent elements as signed integers in the description, but we choose to pack and unpack the serialized *unsigned* integers. The byte string stores unsigned representatives, and unpacking maps them back to centered signed representatives. This avoids implementation issues with sign extension. For a sequence of b -bit coefficients $(w_i)_{0 \leq i < N}$, define

$$\text{Enc}_b(a) = (2^{b-1} - 1 - a) \bmod 2^b \in \{0, \dots, 2^b - 1\}$$

and

$$\text{Pack}((w_i)_{0 \leq i < N}, b) = \text{Enc}_b(w_0) \parallel \dots \parallel \text{Enc}_b(w_{N-1})$$

to be its serialization, where concatenation is defined such that w_0 is stored at the lowest significant address and $\text{Enc}_b(w_0) \bmod 2$ corresponds to bit 0. Unpacking applies the inverse map $\text{Dec}_b(u) = 2^{b-1} - 1 - u$, returning coefficients in $[-2^{b-1}, 2^{b-1} - 1]$. For a ring element $\mathbf{w} = \sum_{i=0}^{k-1} (\sum_{j=0}^{n-1} w_{i,j} y^j) x^i \in \mathbf{S}_{q,n,k}$ with b -bit coefficients, define its serialization to be

$$\text{Pack}(\mathbf{w}, b) = \text{Pack}((w_{0,j})_{0 \leq j < n}, b) \parallel \dots \parallel \text{Pack}((w_{k-1,j})_{0 \leq j < n}, b),$$

again storing $\text{Pack}((w_{0,j})_{0 \leq j < n}, b)$ in the least significant addresses.

- We define the serialization of the public key as

$$\text{pkEncode}(\rho, \mathbf{b}_1) = \rho \parallel \text{Pack}(\mathbf{b}_1, \log_2(p) - d).$$

The corresponding deserialization function is $\text{pkDecode}(pk)$. The total size of the public key is $64 + (\log_2(p) - d)kn/8$ bytes.

- We define the serialization of the private key as

$$\begin{aligned} \text{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{b}_0) = & \rho \parallel K \parallel tr \\ & \parallel \text{Pack}(\mathbf{s}_1, \log_2(2\eta)) \\ & \parallel \text{Pack}(\mathbf{s}_2, \log_2(2\eta)) \\ & \parallel \text{Pack}(\mathbf{b}_0, d). \end{aligned}$$

The corresponding deserialization function is $\text{skDecode}(sk)$. The total size of the private key is $64 + 64 + 128 + (2\log_2(2\eta) + d)kn/8$ bytes.

- The serialization of the signature requires encoding the hint vector, which is the output of the MakeHint_q function, consisting of kn binary coefficient positions, organized as k polynomials with n coefficients each. We follow the approach of Dilithium/ML-DSA [DKL⁺18] to encode the hint as ω total coefficient indices, and k indices to encode when the next polynomial should be selected. As we have $n = 1024$, each coefficient index needs $\log_2(1024) = 10$ bits for its representation. Therefore the total hint size is $10(\omega + k)/8$ bytes, where we choose ω such that $10(\omega + k) \equiv 0 \pmod{8}$, i.e., $(\omega + k) \equiv 0 \pmod{4}$. We call this function hintEncode . The encoding is canonical: within each polynomial the encoded hint indices are strictly increasing, the k delimiter indices are nondecreasing and lie between 0 and ω , and all unused index slots after the last delimiter are zero. The decoding routine sigDecode rejects any signature whose hint encoding is not canonical. We choose ω heuristically so that the chance of generating more than ω hint coefficients is small. We include macros in the reference implementation to validate these choices. The value of ω does not affect the security of the scheme, but reduces the signature size at the cost of moderately slower signing.

Finally, we can define

$$\text{sigEncode}(\tilde{c}, \mathbf{z}, \mathbf{h}) = \tilde{c} \parallel \text{Pack}(\mathbf{z}, \log_2(2\gamma_1)) \parallel \text{hintEncode}(\mathbf{h}).$$

The corresponding deserialization function is $\text{sigDecode}(\sigma)$. The total size of the signature is $\lambda/4 + \log_2(2\gamma_1)kn/8 + 10(\omega + k)/8$ bytes.

Hedged signatures. As is common for modern signatures we allow both randomized and deterministic signatures by including a byte array rnd into the derivation function for the ephemeral randomness. Setting rnd to a random value leads to randomized signatures, fixing it to a known constant will result in a deterministic variant. For settings where physical side channel attacks are a concern, the randomized version is recommended.

Fiat-Shamir with aborts loop. The Fiat-Shamir with aborts loop rejects when any of the following conditions holds: $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$, $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$, $\|c_p^q \mathbf{b}_0\|_\infty \geq \gamma_2$, or $\text{HW}(\mathbf{h}) > \omega$. The first two conditions dominate the rejection probability and can be estimated analytically as follows. One shows easily (e.g., see [LDK⁺22, Section 3.4]) that each coefficient z_i of $\mathbf{z}$ satisfies $|z_i| < \gamma_1 - \beta$ if and only if $-\gamma_1 + \beta - \sigma_i < y_i < \gamma_1 - \beta - \sigma_i$, where y_i and σ_i are the corresponding

coefficients of $\mathbf{y}$ and $c\mathbf{s}_1$, respectively. As each of the kn coefficients of $\mathbf{y}$ is uniformly distributed in $[-\gamma_1, \gamma_1 - 1]$, the probability that $\mathbf{z}$ passes the z -bound test is

$$\Pr[\|\mathbf{z}\|_\infty < \gamma_1 - \beta] = \left(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1} \right)^{kn} = \begin{cases} 0.777 & \text{for Octarine-128,} \\ 0.753 & \text{for Octarine-256,} \\ 0.653 & \text{for Octarine-512.} \end{cases}$$

This probability does not depend on any secret data. Similarly, with the assumption that the centered low bits $\mathbf{r}_0 = \text{LowBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2)$ are uniformly distributed in $[-\gamma_2, \gamma_2 - 1]$, the probability of passing the low-bit test is

$$\Pr[\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta] = \left(\frac{2(\gamma_2 - \beta) - 1}{2\gamma_2} \right)^{kn} = \begin{cases} 0.604 & \text{for Octarine-128,} \\ 0.568 & \text{for Octarine-256,} \\ 0.427 & \text{for Octarine-512.} \end{cases}$$

As the success rate for $\mathbf{r}_0$ is lower than for $\mathbf{z}$, it is more efficient to do the rejection based on $\mathbf{r}_0$ first in the implementation. This way fewer computations have been completed when rejection occurs.

Additionally, we can bound the absolute value of each coefficient of $c\frac{q}{p}\mathbf{b}_0$ by $\frac{q}{p}\tau 2^{d-1}$, since c contains exactly τ non-zero elements that are either 1 or -1 , and each coefficient of $\mathbf{b}_0$ lies in $[-2^{d-1}, 2^{d-1} - 1]$. This is a conservative coefficient-wise bound used in the correctness and security estimates; for the parameter sets in Table 2, the bound equals γ_2 for Octarine-128 and exceeds γ_2 for Octarine-256 and Octarine-512. This worst-case bound is nevertheless loose: it corresponds to all τ signed contributions aligning in the same direction, whereas typical keys and challenges produce substantial cancellation in the sparse product. We therefore check the contribution of the $c\frac{q}{p}\mathbf{b}_0$ -bound and hint-weight tests empirically rather than inferring it from this deterministic bound alone. In a run of the reference implementation generating 5000 signatures each for Octarine-128, Octarine-256, and Octarine-512, the numbers of rejected iterations caused by the $c\frac{q}{p}\mathbf{b}_0$ -bound test were 0, 30, and 1, respectively, and the corresponding numbers caused by the hint-weight test were 36, 32, and 8. The corresponding average numbers of loop iterations, counting all rejection tests, were 2.10, 2.29, and 3.54, respectively. The lower count for Octarine-512 than for Octarine-256 reflects that these empirical tails are governed by the typical cancellation scale and by the chosen γ_2 and ω , not by the linear worst-case bound alone. In Table 2 we use the product

$$\frac{1}{\Pr[\|\mathbf{z}\|_\infty < \gamma_1 - \beta] \cdot \Pr[\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta]}$$

as an analytic approximation of the number of rounds required for completing the signing on average. The empirical averages above are close to, and in this run slightly below, this two-test estimate; the small residual rejection counts are within the variation introduced by the heuristic uniformity assumptions and finite measurement sample.

Challenge generation. The function $\text{SampleInBall}(\tilde{c})$ deterministically maps a hash digest $\tilde{c}$ to a challenge polynomial $c \in \mathbf{R}_{q,n}$ with exactly τ non-zero coefficients, each in $\{-1, +1\}$. The seed $\tilde{c}$ is expanded with XOF_C . The sign-bit stream uses stream identifier 0 and counter 0; the index streams use stream identifier 1 and a 32-bit counter. The algorithm is an inside-out Fisher–Yates shuffle applied to the n integer coefficients of a polynomial in $\mathbf{R}_{q,n}$. Since $n = 1024$, each candidate shuffle index is read as a 10-bit word from the XOF stream and rejected unless it lies in $\{0, \dots, i\}$. We refer to Algorithm 2 for the full description.

The challenge entropy is $\log_2 \binom{n}{\tau} + \tau$ bits. This is the entropy of the uniform distribution over all signed Hamming-weight- τ challenges. As we choose $n = 1024$, we can select smaller τ

Algorithm 2 Sampling a challenge polynomial.

SampleInBall($\tilde{c}$)

```

1:  $c = (c_0, c_1, \dots, c_{n-1}) \leftarrow \mathbf{0}$ 
2:  $\text{idx\_ctr} \leftarrow 0$ 
3:  $(s_0, \dots, s_{\tau-1}) \leftarrow \text{XOF}_C(\tilde{c}, \text{IntegerToBytes}(0, 1), \text{IntegerToBytes}(0, 4); \lceil \tau/8 \rceil)$   $\triangleright$  Sample the
   sign bits
4:  $(B_0, \dots, B_{79}) \leftarrow \text{XOF}_C(\tilde{c}, \text{IntegerToBytes}(1, 1), \text{IntegerToBytes}(\text{idx\_ctr}, 4); 80)$   $\triangleright$  Sample
   byte buffer for 64 indices
5:  $\text{bitpos} \leftarrow 0$ 
6: for  $i = n - \tau$  to  $n - 1$  do
7:   do
8:     if  $\text{bitpos} + 10 > 640$  then
9:        $\text{idx\_ctr} \leftarrow \text{idx\_ctr} + 1$ 
10:       $(B_0, \dots, B_{79}) \leftarrow \text{XOF}_C(\tilde{c}, \text{IntegerToBytes}(1, 1), \text{IntegerToBytes}(\text{idx\_ctr}, 4); 80)$ 
11:       $\text{bitpos} \leftarrow 0$ 
12:    end if
13:     $j \leftarrow 0$ 
14:    for  $\ell = 0$  to  $9$  do
15:       $j \leftarrow j + 2^\ell \cdot \text{bit}(B_0, \dots, B_{79}; \text{bitpos})$ 
16:       $\text{bitpos} \leftarrow \text{bitpos} + 1$ 
17:    end for
18:    while  $(j > i)$ 
19:       $c_i \leftarrow c_j$ 
20:       $c_j \leftarrow (-1)^{s_{i-(n-\tau)}}$ 
21:    end while
22: return  $c$   $\triangleright$  Interpret as an element of  $\mathbf{R}_{q,n}$ 

```

* The output has exactly τ non-zero coefficients, each in $\{-1, +1\}$. The algorithm is an inside-out Fisher–Yates shuffle applied to the n integer coefficients of a polynomial in $\mathbf{R}_{q,n}$. Since $n = 1024$, shuffle indices are sampled as 10-bit words.

than Module-LWE proposals with $n = 256$, allowing to reduce the size of β and reducing the number of aborts in the signing loop. We select τ to be the smallest value such that the exact challenge entropy is at least λ ; Table 2 reports the corresponding entropy values rounded down to whole bits.

In the signature algorithms, c is embedded into $\mathbf{S}_{q,n,k}$ as the constant component, i.e., $c := c(y) + \sum_{i=1}^{k-1} 0 \cdot x^i$. This mimics the case of M-LWE where c is a single ring element, not a full module element. Thus, for $\mathbf{s} = \sum_{i=0}^{k-1} s_i x^i$, multiplication by c acts component-wise as $c\mathbf{s} = \sum_{i=0}^{k-1} (cs_i)x^i$.

Verification conditions. We sketch why an honestly generated signature always passes verification. Recall from KeyGen (see Algorithm 3) that $\mathbf{b} = \lfloor \frac{p}{q} \mathbf{A} \mathbf{s}_1 \rfloor \in \mathbf{S}_{p,n,k}$ is the LWR public key, $\mathbf{s}_2 = \mathbf{A} \mathbf{s}_1 - \frac{q}{p} \mathbf{b}$ is the rounding error (so that $\mathbf{A} \mathbf{s}_1 - \mathbf{s}_2 = \frac{q}{p} \mathbf{b}$), and $(\mathbf{b}_1, \mathbf{b}_0) = \text{Power2Round}_p(\mathbf{b}, 2^d)$.

During signing, the signer computes $\mathbf{w} = \mathbf{A} \mathbf{y}$ and $\mathbf{r}_0 = \text{LowBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2)$. The rejection condition ensures $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$. Since $\|c\mathbf{s}_2\|_\infty \leq \beta$ by definition of β , the addition of $c\mathbf{s}_2$ could not have caused carries in the high bits of $\mathbf{w}$. Therefore it follows that $\text{HighBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2) = \text{HighBits}_q(\mathbf{w}, 2\gamma_2)$.

During verification, the verifier computes

$$\begin{aligned}
\mathbf{w}'_{\text{Approx}} &= \mathbf{A}\mathbf{z} - c_p^q \mathbf{b}_1 \cdot 2^d \\
&= \mathbf{A}(\mathbf{y} + c\mathbf{s}_1) - c_p^q (\mathbf{b} - \mathbf{b}_0) \\
&= \mathbf{A}\mathbf{y} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0 \\
&= \mathbf{w} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0,
\end{aligned}$$

using that $\mathbf{A}\mathbf{s}_1 - \mathbf{s}_2 = \frac{q}{p}\mathbf{b}$ and the Power2Round_p decomposition $\mathbf{b} = \mathbf{b}_1 \cdot 2^d + \mathbf{b}_0$. The hint $\mathbf{h} = \text{MakeHint}_q(-c_p^q \mathbf{b}_0, \mathbf{w} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0, 2\gamma_2)$ encodes the carry positions where $c_p^q \mathbf{b}_0$ affects the high bits. Hence, $\text{UseHint}_q(\mathbf{h}, \mathbf{w}'_{\text{Approx}}, 2\gamma_2) = \text{HighBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2) = \mathbf{w}_1$, and the verifier correctly recovers $\mathbf{w}_1$ and can compare its digest against the one received in the signature.

To ensure no malicious signatures are accepted, the verifier must check that $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$. Allowing larger $\mathbf{z}$ will decrease the difficulty of the forgery, affecting the target security level. The explicit check $\text{HW}(\mathbf{h}) > \omega$ is redundant if sigDecode enforces canonical hint encoding with at most ω indices, but we keep it in Verify for clarity.

Algorithm 3 Octarine KeyGen, Sign, and Verify algorithms.

$(pk, sk) \leftarrow \text{Octarine.KeyGen}()$

- 1: $\xi \leftarrow U(\mathbb{B}^{64})$ $\triangleright$ Sample initial seed
- 2: $(\rho, \rho', K) \in \mathbb{B}^{64} \times \mathbb{B}^{128} \times \mathbb{B}^{64} \leftarrow H_{\text{kdf}}(\xi; |\rho| + |\rho'| + |K|)$
- 3: $\mathbf{A} \leftarrow D_{\mathbf{A}}(\mathbf{S}_{q,n,k}; \rho)$
- 4: $\mathbf{s}_1 \leftarrow D_{\mathbf{s}}(\mathbf{S}_{q,n,k}; \rho')$
- 5: $\mathbf{t} = \mathbf{A}\mathbf{s}_1 \in \mathbf{S}_{q,n,k}$
- 6: $\mathbf{b} = \lfloor \frac{q}{p} \mathbf{t} \rfloor \in \mathbf{S}_{p,n,k}$ $\triangleright$ Rounded public value in $\mathbf{S}_{p,n,k}$
- 7: $\mathbf{s}_2 = \mathbf{t} - \frac{q}{p} \mathbf{b} \in \mathbf{S}_{q/p,n,k}$ $\triangleright \frac{q}{p} \mathbf{b} = \mathbf{A}\mathbf{s}_1 - \mathbf{s}_2$, with derived $\mathbf{s}_2$ bounded in $[-\frac{q}{2p}, \frac{q}{2p} - 1]$
- 8: $(\mathbf{b}_1, \mathbf{b}_0) \leftarrow \text{Power2Round}_p(\mathbf{b}, 2^d)$ $\triangleright$ Compress $\mathbf{b} \in S_p$
- 9: $pk \leftarrow \text{pkEncode}(\rho, \mathbf{b}_1)$
- 10: $tr \leftarrow H_{\text{tr}}(pk; |tr|)$
- 11: $sk \leftarrow \text{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{b}_0)$ $\triangleright K$ and tr are for use in signing
- 12: **return** (pk, sk)

$\sigma \leftarrow \text{Octarine.Sign}(sk, M, rnd)$

- 1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{b}_0) \leftarrow \text{skDecode}(sk)$
- 2: $\mathbf{A} \leftarrow D_{\mathbf{A}}(\mathbf{S}_{q,n,k}; \rho)$
- 3: $\mu \leftarrow H_{\text{mu}}(tr, M; |\mu|)$
- 4: $\rho'' \leftarrow H_{\text{rhopp}}(K, rnd, \mu; |\rho''|)$
- 5: $\kappa \leftarrow 0$
- 6: $(z, h) \leftarrow \perp$
- 7: **while** $(z, h) = \perp$ **do**
- 8: $\mathbf{y} \leftarrow D_{\mathbf{y}}(\mathbf{S}_{q,n,k}; \rho'', \kappa)$
- 9: $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$
- 10: $\mathbf{w}_1 = \text{HighBits}_q(\mathbf{w}, 2\gamma_2)$
- 11: $\tilde{c} = H_{\text{ch}}(\mu, \mathbf{w}_1; |\tilde{c}|)$
- 12: $c = \text{SampleInBall}(\tilde{c})$
- 13: $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$
- 14: $\mathbf{r}_0 = \text{LowBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2)$
- 15: **if** $\|\mathbf{r}_0\|_{\infty} \geq \gamma_2 - \beta$ **or** $\|\mathbf{z}\|_{\infty} \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
- 16: **else**
- 17: $\mathbf{h} = \text{MakeHint}_q(-c\frac{q}{p}\mathbf{b}_0, \mathbf{w} + c\mathbf{s}_2 + c\frac{q}{p}\mathbf{b}_0, 2\gamma_2)$
- 18: **if** $\|c\frac{q}{p}\mathbf{b}_0\|_{\infty} \geq \gamma_2$ **or** $\text{HW}(\mathbf{h}) > \omega$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
- 19: **end if**
- 20: **end if**
- 21: $\kappa = \kappa + k$
- 22: **end while**
- 23: **return** $\sigma = \text{sigEncode}(\tilde{c}, \mathbf{z}, \mathbf{h})$

$\text{bool} \leftarrow \text{Octarine.Verify}(pk, M, \sigma)$

- 1: $(\rho, \mathbf{b}_1) \leftarrow \text{pkDecode}(pk)$
 - 2: $(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow \text{sigDecode}(\sigma)$
 - 3: $\mathbf{A} \leftarrow D_{\mathbf{A}}(\mathbf{S}_{q,n,k}; \rho)$
 - 4: $tr \leftarrow H_{\text{tr}}(pk; |tr|)$
 - 5: $\mu \leftarrow H_{\text{mu}}(tr, M; |\mu|)$
 - 6: $c \leftarrow \text{SampleInBall}(\tilde{c})$
 - 7: $\mathbf{w}'_{\text{Approx}} \leftarrow \mathbf{A}\mathbf{z} - c\frac{q}{p}\mathbf{b}_1 2^d$
 - 8: $\mathbf{w}'_1 \leftarrow \text{UseHint}_q(\mathbf{h}, \mathbf{w}'_{\text{Approx}}, 2\gamma_2)$
 - 9: $\tilde{c}' \leftarrow H_{\text{ch}}(\mu, \mathbf{w}'_1; |\tilde{c}'|)$
 - 10: **return** $[\|\mathbf{z}\|_{\infty} < \gamma_1 - \beta]$ **and** $[\tilde{c} = \tilde{c}']$ **and** $[\text{HW}(\mathbf{h}) \leq \omega]$
-

5 Design Rationale

SIS-type assumption. The unforgeability of the scheme additionally requires a Module-SIS-type assumption over $\mathbf{S}_{q,n,k}$. Specifically, one needs the hardness of finding a short non-zero vector $\mathbf{z} \in \mathbf{S}_{q,n,k}^\ell$ such that $\mathbf{A}\mathbf{z} = \mathbf{0}$ in $\mathbf{S}_{q,n,k}$. Both the RR-LWR and SIS-type assumptions can be analyzed via standard lattice reduction attacks (see Section 6).

5.1 Criterion for Parameter Selection

The parameters of the scheme are $q, p, n, k, \gamma_1, \gamma_2, \beta$, and ω . Their roles are:

- q : the main modulus (power of two), controls the size of ring elements and the lattice dimension for security.
- p : the LWR rounding modulus (power of two, $p < q$), governs the public key compression of $\mathbf{b}$.
- n, k : n is the base-ring degree and k is the radical extension degree. Together they determine the total algebraic degree nk , which is the effective lattice dimension used in the generic attack estimates. Increasing k increases the lattice dimension and thus the security level, at the cost of larger keys and signatures.
- γ_1 : the masking range, must be large enough so that rejection sampling does not reveal information about $\mathbf{s}_1$, yet small enough to keep signature sizes manageable.
- γ_2 : the low-order rounding range (power of two), determines the granularity of the HighBits decomposition and the number of hint bits. A smaller γ_2 reduces the hint size but increases the rejection probability.
- β : the rejection bound, set as $\beta = \tau \cdot \eta$ where τ is the Hamming weight of the challenge polynomial and η bounds the coefficients of $\mathbf{s}_1$.
- ω : the maximum Hamming weight of the hint vector $\mathbf{h}$.

The parameters are selected to simultaneously satisfy:

1. *Correctness*: The rejection conditions $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$ ensure that all valid signatures pass verification.
2. *Security*: The hardness of the RR-LWR and SIS-type problems at the target security level must hold for the chosen lattice dimensions and moduli. Concrete estimates are provided in Section 6.
3. *Efficiency*: The expected number of signing repetitions (rejection rate) should be small, and the resulting key and signature sizes should be compact.

5.2 Correctness and Rejection-Probability Analysis

The Fiat–Shamir with Aborts paradigm guarantees *perfect correctness*: every signature that survives the rejection conditions in the Sign algorithm is accepted by the Verify algorithm with probability 1. In other words, there is no analogue of a “decryption failure” event.

Lemma 1 (High-bit stability). *Let $m = 2\gamma_2$ divide q , and let HighBits_q and LowBits_q be defined coefficient-wise with low bits represented in $[-\gamma_2, \gamma_2 - 1]$. For ring elements $a, e \in \mathbf{S}_{q,n,k}$, suppose*

$$r_0 = \text{LowBits}_q(a + e, m), \quad \|r_0\|_\infty < \gamma_2 - \beta, \quad \|e\|_\infty \leq \beta.$$

Then

$$\text{HighBits}_q(a + e, m) = \text{HighBits}_q(a, m).$$

Proof. It suffices to argue coefficient-wise. Write

$$a + e = hm + r_0 \pmod{q}$$

with $r_0 \in [-\gamma_2, \gamma_2 - 1]$. The assumptions imply that $r_0 - e$ still lies in the same low-bit interval. Hence

$$a = hm + (r_0 - e) \pmod{q}$$

has the same high-bit component h as $a + e$. $\square$

Lemma 2 (Hint correctness). *Let $m = 2\gamma_2$ divide q , and let MakeHint_q and UseHint_q be the coefficient-wise algorithms from Algorithm 1. For ring elements $r, e \in \mathbf{S}_{q,n,k}$, suppose $\|e\|_\infty < \gamma_2$. If*

$$\mathbf{h} = \text{MakeHint}_q(e, r, m),$$

then

$$\text{UseHint}_q(\mathbf{h}, r, m) = \text{HighBits}_q(r + e, m).$$

Proof. Again the statement is coefficient-wise. Write $r = r_1m + r_0 \pmod{q}$, with $r_0 \in [-\gamma_2, \gamma_2 - 1]$. If $\text{HighBits}_q(r, m) = \text{HighBits}_q(r + e, m)$, then MakeHint_q returns 0 and UseHint_q returns r_1 . Otherwise, since $|e| < \gamma_2 = m/2$, the perturbation can cross at most one decomposition boundary. If $r_0 > 0$, the only possible crossing is upward, and UseHint_q returns $r_1 + 1$. If $r_0 \leq 0$, the only possible crossing is downward, and UseHint_q returns $r_1 - 1$. These are exactly the high bits of $r + e$, modulo q/m . $\square$

This is ensured by the four rejection conditions in the signing loop:

1. The condition $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ ensures that the response vector $\mathbf{z}$ remains within the range accepted by the verifier.
2. The condition $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$, where $\mathbf{r}_0 = \text{LowBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2)$, ensures that the low-order bits are small enough so that $\text{HighBits}_q(\mathbf{w} + c\mathbf{s}_2, 2\gamma_2) = \text{HighBits}_q(\mathbf{w}, 2\gamma_2)$ by Lemma 1.
3. The condition $\|c_p^q \mathbf{b}_0\|_\infty < \gamma_2$ ensures that the public-key-compression perturbation is small enough for the coefficient-wise $\text{MakeHint}_q / \text{UseHint}_q$ carry correction to be valid. In the honest signing equation we apply Lemma 2 with the substitution $r = \mathbf{w} + c\mathbf{s}_2 + c_p^q \mathbf{b}_0$ and $e = -c_p^q \mathbf{b}_0$.
4. The hint weight condition $\text{HW}(\mathbf{h}) \leq \omega$ bounds the number of carry corrections, ensuring that the hint can be represented within the allocated space in the signature.

When all four conditions are met, the correctness argument in Section 4 shows that the verifier recovers $\mathbf{w}_1$ exactly.

The cost of this approach is that multiple loop iterations may be required before a valid signature is produced. The expected number of repetitions depends on the ratio of the masking range γ_1 to the rejection bound β and on the rounding parameter γ_2 . Our parameters are chosen so that the expected number of repetitions remains small (below 4) for all security levels. Concrete values are listed in Table 2.

5.3 Other Considerations

Power-of-two modulus q . The choice of a power-of-two modulus q (as opposed to a prime modulus as in Dilithium) brings several benefits:

- *Simpler rounding:* All rounding operations (Power2Round , HighBits , LowBits) reduce to bit-shift operations, eliminating the need for modular reduction routines for these steps.

- *Compatibility with the KEM*: The KEM uses the same power-of-two modulus q , so NTT routines, CRT representations, and sampling infrastructure are shared between the two schemes.
- *Side-channel and fault protection*: Operations involving powers of two are inherently branch-free and have uniform timing, which aids constant-time implementation. Furthermore, the power-of-two structure makes certain fault-injection attack scenarios easier to reason about.
- *Simpler security argument*: With a power-of-two q , the RR-LWR assumption does not require the technical “modulus switching” argument needed when q is prime [BPR12], as the rounding is directly a truncation of bits.

Challenge space and entropy. The challenge is sampled as a sparse polynomial $c \in \mathbf{R}_{q,n}$ with exactly τ non-zero coefficients in $\{-1, +1\}$, and is embedded into $\mathbf{S}_{q,n,k}$ as the constant x^0 component. For $\mathbf{s} = \sum_{i=0}^{k-1} s_i x^i$, multiplication by c is therefore component-wise:

$$c\mathbf{s} = \sum_{i=0}^{k-1} (cs_i)x^i.$$

For the uniform distribution over all such signed Hamming-weight- τ challenges, the challenge entropy is $\log_2 \binom{n}{\tau} + \tau$ bits. The SampleInBall procedure in Algorithm 2 implements this distribution by rejection sampling 10-bit shuffle indices for $n = 1024$.

6 Security Statements and Analyses

6.1 Theoretical Security

6.1.1 RR-LWR hard problem

Let n be a power of two and k a strictly positive integer. It can be shown that the polynomial $f(x) = (x^k - 2)^n + 1$ is irreducible [BRVW26, Lemma 1], implying that $\mathbf{Q}[x]/(f(x))$ is a number field and as such we can define a Ring-LWR problem over this number field. This is the number-field analogue of standard LWR [BPR12], instantiated in the radical-ring representation used by Octarine.

From the polynomial $f(x) = (x^k - 2)^n + 1$, we define the RR-LWR problem over the ring of integers $\mathcal{O}_{\mathbf{K}}$ of the number field $\mathbf{K} = \mathbf{Q}[x]/(f(x)) = \mathbf{Q}[\alpha]$. In contrast to Ring-LWE, an LWR sample does not contain an independently sampled Gaussian error term; the noise is the deterministic quantization error introduced by the coefficient-wise map

$$z \in \mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}} \mapsto \left\lfloor \frac{p}{q}z \right\rfloor \in \mathcal{O}_{\mathbf{K}}/p\mathcal{O}_{\mathbf{K}}.$$

Via an explicit computation (e.g. using Dedekind’s criterion) one can verify that $\mathcal{O}_{\mathbf{K}} = \mathbf{Z}[\alpha]$. Thus the implementation ring

$$S_{q,n,k} = (\mathbf{Z}_q[y]/(y^n + 1)[x]) / (x^k - y - 2)$$

is simply a concrete coefficient representation of $\mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}$.

Definition 2 (Search RR-LWR). *Let $q > p \geq 2$, let*

$$S_q = S_{q,n,k} = (\mathbf{Z}_q[y]/(y^n + 1)[x]) / (x^k - y - 2), \quad S_p = S_{p,n,k},$$

and let $D_{\mathbf{s}}$ be a secret distribution over S_q . For a fixed secret $\mathbf{s} \in S_q$, define the **RR-LWR** sample distribution $\mathcal{L}_{\mathbf{s},q,p}^{\text{rr-lwr}}$ over $S_q \times S_p$ by sampling $\mathbf{A} \leftarrow S_q$ uniformly and outputting

$$\left(\mathbf{A}, \left\lfloor \frac{p}{q} \mathbf{A} \mathbf{s} \right\rfloor \right),$$

where rounding is applied coefficient-wise in the $y^i x^j$ basis. The search **RR-LWR** problem is: given N_{samp} independent samples from $\mathcal{L}_{\mathbf{s},q,p}^{\text{rr-lwr}}$, for $\mathbf{s} \leftarrow D_{\mathbf{s}}$, recover $\mathbf{s}$. The advantage of an algorithm $\mathcal{A}$ is

$$\text{Adv}_{q,p,D_{\mathbf{s}},N_{\text{samp}}}^{\text{s-rr-lwr}}(\mathcal{A}) = \Pr[\mathcal{A}((\mathbf{A}_i, \mathbf{b}_i)_{i=1}^{N_{\text{samp}}}) = \mathbf{s}].$$

Definition 3 (Decision **RR-LWR**). With notation as in Definition 2, the decision **RR-LWR** problem is to distinguish the following two experiments:

$$\begin{aligned} \text{Real} : & \quad \mathbf{s} \leftarrow D_{\mathbf{s}}, \quad (\mathbf{A}_i, \mathbf{b}_i) \leftarrow \mathcal{L}_{\mathbf{s},q,p}^{\text{rr-lwr}} \quad \text{for } i = 1, \dots, N_{\text{samp}}, \\ \text{Rand} : & \quad \mathbf{A}_i \leftarrow S_q, \quad \mathbf{b}_i \leftarrow S_p \quad \text{uniformly and independently for } i = 1, \dots, N_{\text{samp}}. \end{aligned}$$

The advantage of a distinguisher $\mathcal{A}$ is

$$\text{Adv}_{q,p,D_{\mathbf{s}},N_{\text{samp}}}^{\text{d-rr-lwr}}(\mathcal{A}) = |\Pr[\mathcal{A}(\text{Real}) = 1] - \Pr[\mathcal{A}(\text{Rand}) = 1]|.$$

The decision **RR-LWR** assumption states that this advantage is negligible for all efficient distinguishers $\mathcal{A}$.

Definition 4 (Decision key-generation **RR-LWR**). With notation as in Definition 2, the key-generation decision **RR-LWR** problem is the one-sample decision problem for the expanded public-key view of the signature key-generation algorithm in Algorithm 3, after the seeded public matrix has been replaced by an explicit uniform element. A challenger outputs a pair in $S_q \times S_p$. In the real experiment the pair is

$$\text{KGReal}_{q,p,D_{\mathbf{s}}} = \left(\mathbf{A}, \left\lfloor \frac{p}{q} \mathbf{A} \mathbf{s} \right\rfloor \right), \quad \mathbf{A} \leftarrow S_q, \quad \mathbf{s} \leftarrow D_{\mathbf{s}},$$

and the random distribution is

$$\text{KGRand}_{q,p} = (\mathbf{A}, \mathbf{u}), \quad \mathbf{A} \leftarrow S_q, \quad \mathbf{u} \leftarrow S_p,$$

with $\mathbf{A}$ and $\mathbf{u}$ sampled uniformly and independently. The distinguisher receives only the displayed pair, and in particular does not receive the secret $\mathbf{s}$. The advantage of a distinguisher $\mathcal{A}$ is

$$\text{Adv}_{\text{kg},q,p,D_{\mathbf{s}}}^{\text{d-rr-lwr}}(\mathcal{A}) = |\Pr[\mathcal{A}(\text{KGReal}_{q,p,D_{\mathbf{s}}}) = 1] - \Pr[\mathcal{A}(\text{KGRand}_{q,p}) = 1]|.$$

The following theorem states the representation reduction formally.

Theorem 3 (RR-LWR as transported-basis Ring-LWR over the radical number field). Let $q > p \geq 2$ with $p \mid q$, let $f(x) = (x^k - 2)^n + 1$ be irreducible, and let $\mathbf{K} = \mathbf{Q}[x]/(f(x)) = \mathbf{Q}[\alpha]$ with $\mathcal{O}_{\mathbf{K}} = \mathbf{Z}[\alpha]$. Let

$$\varphi_q : S_{q,n,k} \longrightarrow \mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}$$

be the ring isomorphism induced by $x \mapsto \alpha$ and $y \mapsto \alpha^k - 2$, and define φ_p analogously modulo p . Let $\mathcal{B} = \{(\alpha^k - 2)^i \alpha^j : 0 \leq i < n, 0 \leq j < k\}$ be the $\mathbf{Z}$ -basis of $\mathcal{O}_{\mathbf{K}}$ which is the image of the monomial basis $\{y^i x^j\}$ under φ_q , and let $\lfloor \cdot \rfloor_{\mathcal{B}}$ denote rounding from $\mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}$ to $\mathcal{O}_{\mathbf{K}}/p\mathcal{O}_{\mathbf{K}}$ applied coefficient-wise in $\mathcal{B}$. For any secret distribution $D_{\mathbf{s}}$ over $S_{q,n,k}$, any distinguisher $\mathcal{A}$ for **RR-LWR** samples

$$\left(\mathbf{A}, \left\lfloor \frac{p}{q} \mathbf{A} \mathbf{s} \right\rfloor \right) \in S_{q,n,k} \times S_{p,n,k}, \quad \mathbf{A} \leftarrow S_{q,n,k}, \quad \mathbf{s} \leftarrow D_{\mathbf{s}},$$

against uniform samples yields a distinguisher $\mathcal{B}'$ for the corresponding decision Ring-LWR problem over $\mathcal{O}_{\mathbf{K}}$ with rounding $\lfloor \cdot \rfloor_{\mathcal{B}}$, secret distribution $\varphi_q(D_{\mathbf{s}})$, the same number of samples, essentially the same running time, and the same advantage.

Proof. The maps φ_q and φ_p are efficiently computable bijections because $S_{q,n,k} \cong \mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}$ and $S_{p,n,k} \cong \mathcal{O}_{\mathbf{K}}/p\mathcal{O}_{\mathbf{K}}$ (the latter using $\mathcal{O}_{\mathbf{K}} = \mathbf{Z}[\alpha]$). The rounding operation used in RR-LWR is coefficient-wise in the basis $\{y^i x^j\}$, and φ_q maps this basis to $\mathcal{B}$; thus by definition of $\lfloor \cdot \rfloor_{\mathcal{B}}$, for every $z \in S_{q,n,k}$,

$$\varphi_p\left(\left\lfloor \frac{p}{q} z \right\rfloor\right) = \left\lfloor \frac{p}{q} \varphi_q(z) \right\rfloor_{\mathcal{B}}.$$

The reduction $\mathcal{B}'$ receives Ring-LWR challenge samples $(\tilde{\mathbf{A}}_i, \tilde{\mathbf{b}}_i) \in (\mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}) \times (\mathcal{O}_{\mathbf{K}}/p\mathcal{O}_{\mathbf{K}})$ and maps them to

$$(\mathbf{A}_i, \mathbf{b}_i) = (\varphi_q^{-1}(\tilde{\mathbf{A}}_i), \varphi_p^{-1}(\tilde{\mathbf{b}}_i)).$$

If the challenge samples are Ring-LWR samples with secret $\tilde{\mathbf{s}} = \varphi_q(\mathbf{s})$, then the mapped pairs are exactly RR-LWR samples with secret $\mathbf{s}$; if the challenges are uniform, the mapped pairs are uniform because φ_q, φ_p are bijections. Hence $\mathcal{B}'$ has the same advantage as $\mathcal{A}$. $\square$

We stress that Theorem 3 reduces RR-LWR to a Ring-LWR variant whose rounding is coefficient-wise in the transported monomial basis $\mathcal{B}$ rather than in the canonical embedding. For the concrete estimates, we account for the corresponding basis distortion through the linear basis change L discussed below: its singular values $\sigma_{\min}(L) = \sqrt{nk}$ and $\sigma_{\max}(L) \leq C_{\text{emb}}\sqrt{nk}$ (with the small constants C_{emb} tabulated below) show that canonical-embedding norm estimates should be interpreted with at most a factor C_{emb} distortion in the relevant secret/error norms.

From f it is easy to explicitly write down the nk roots as

$$r_{i,j} = (\xi_{2n}^{(2j+1)} + 2)^{(1/k)} \xi_k^i$$

for $i = 0, \dots, k-1$ and $j = 0, \dots, n-1$, where ξ_m is a primitive m -th root of unity. Define an ordering of these roots such that $r_{m+nk/2} = \bar{r}_m$ for $m = 1, \dots, nk/2$. Then the canonical embedding of an element $a(x) \in \mathbf{K}$ consists of the nk evaluations $a(r_i)$ for $i = 1, \dots, nk$.

To compare our RR-LWR instance with the usual canonical-embedding viewpoint, it remains to account for the difference between sampling coefficient-wise in the basis $y^i x^j$ for $i = 0, \dots, n-1$ and $j = 0, \dots, k-1$ in our ring representation $(\mathbf{Z}_q[y]/(y^n + 1)[x])/(x^k - y - 2)$ and the canonical embedding of $\mathbf{K}$. Note that this is a linear transformation L that can be obtained as the composition of two linear transformations:

- the first transformation maps the basis $y^i x^j$ into the corresponding element $b(x) = (x^k - 2)^i x^j \bmod f(x)$ (recall that indeed $y = x^k - 2$ in our ring)
- the second transformation is then the standard canonical embedding transformation mapping $b(x)$ into $[b(r_i)]_{i=1, \dots, nk}$. Note that this transformation is simply given by the Vandermonde matrix whose i -th row consists of the powers of r_i^j for $j = 0, \dots, nk-1$.

To bound the maximum distortion introduced by sampling in the basis $y^i x^j$ vs. in the canonical basis, it suffices to look at the minimum and maximum singular value of the above transformation L . Recall that in the "ideal case" of 2-power cyclotomic $x^n + 1$, the minimum and maximum singular value are the same, namely $\sqrt{n}$. We have computed the singular values of the transformation L for $k = 2, \dots, 15$ and for varying n and we find that the smallest singular value is $\sqrt{nk}$ (note that nk is the extension degree so this is optimal) and the largest singular value is a small multiple $C_{\text{emb}}\sqrt{nk}$ depending on k : for $k = 2$, $C_{\text{emb}} < 1.74$, for $k = 3$, $C_{\text{emb}} < 2.08$, for $k = 5$, $C_{\text{emb}} < 2.41$, for $k = 7$, $C_{\text{emb}} < 2.57$ and for $k = 15$, $C_{\text{emb}} < 2.79$. This shows that sampling in the coefficient basis only incurs a minimal distortion vs. sampling in the canonical basis. As such the secret distribution, expressed in the transported monomial basis used by Octarine, can be compared with the usual canonical-embedding Ring-LWR viewpoint over a degree- kn number field with only this bounded basis distortion. We can therefore estimate the hardness of the resulting RR-LWR instance using generic lattice attacks in dimension kn .

Finally, we note that since $f(x)$ is irreducible, the ring structure is not that of the multivariate rings considered in the m-RLWE literature [PUTPPG15] and a sequence of follow-up papers [PUTPPG16, APUPG17, PUTPG⁺21]. These works consider rings of the form $R_q[x, y] = \mathbf{Z}_q[x, y]/(g(x), h(y))$ where f and g are univariate polynomials defined over $\mathbf{Z}$. The m-RLWE problem was broken for certain choices of g, h in a paper from Bootland, Castryck and Vercauteren [BCV20], in particular when the roots of both g and h lie in a number field whose extension degree is smaller than $\deg(g)\deg(h)$. As the ring modulus $f(x) = (x^k - 2)^n + 1$ from Equation (5) really defines a number field of degree kn , these attacks do not apply. While we use the isomorphism from Equation (5) to make effective use of the ring structure, the lattice problem is a transported-basis Ring-LWR problem over the *univariate* ring $S_{q,n,k}$.

The above discussion gives the RR-LWR hardness assumption used in the signature proof below. Only the key-generation distribution is needed for the signature scheme: there is no encryption challenge in the EUF-CMA or SUF-CMA experiments. The proof works in the expanded public-key view $(\mathbf{A}, \mathbf{b}_1)$, where $\mathbf{A}$ is available explicitly and uniformly. The underlying RR-LWR challenge is still the full pair $(\mathbf{A}, \mathbf{b})$; the reduction computes $(\mathbf{b}_1, \mathbf{b}_0) = \text{Power2Round}_p(\mathbf{b}, 2^d)$, gives only $\mathbf{b}_1$ to the adversary, and keeps $\mathbf{b}_0$ for extraction. The actual public key stores only the seed ρ and regenerates $\mathbf{A} = D_{\mathbf{A}}(\mathbf{S}_{q,n,k}; \rho)$; replacing this seeded sampler by the expanded uniform view costs at most the sampler distinguishing term $\Delta_{\mathbf{A}}$, which is zero when $D_{\mathbf{A}}$ is modeled as an ideal random oracle.

6.1.2 Unforgeability of Octarine

We analyze the signature scheme in the random oracle model. All inputs to the random oracle are encoded with fixed domain-separation labels and length-delimited strings, so that the calls for key derivation, message representatives, signing randomness, and Fiat–Shamir challenges are disjoint random-oracle domains. The adversary is given a public key and oracle access to the signing algorithm. For each signing query, the randomized signing oracle samples a fresh value rnd internally and returns $\text{Octarine.Sign}(sk, M, rnd)$. The deterministic fixed- rnd variant has a different repeated-query transcript distribution and is not the oracle interface used in the proof below. In the EUF-CMA experiment the adversary wins if it outputs a valid pair (M^*, σ^*) for a message M^* that was never submitted to the signing oracle. In the SUF-CMA experiment the adversary wins if it outputs a valid pair (M^*, σ^*) that is different from every pair returned by the signing oracle. Thus SUF-CMA also rules out producing a second valid signature for a previously signed message.

Let $S_q = \mathbf{S}_{q,n,k}$, and let all norms be coefficient-wise infinity norms in the signed coefficient representation. Let

$$\mathcal{C}_\tau = \{c \in \mathbf{R}_{q,n} : c \text{ has exactly } \tau \text{ coefficients in } \{-1, +1\}\}$$

be embedded into S_q through the constant x^0 component, as in Section 4. For a key pair, write

$$\mathbf{b} = \left\lfloor \frac{p}{q} \mathbf{A} \mathbf{s}_1 \right\rfloor, \quad \mathbf{b} = \mathbf{b}_1 2^d + \mathbf{b}_0, \quad \mathbf{A} \mathbf{s}_1 - \mathbf{s}_2 = \frac{q}{p} \mathbf{b}.$$

RR-MSIS relations. The SIS-type assumption used for unforgeability is the radical-ring analogue of the module short integer solution problem, which we denote by RR-MSIS. In its homogeneous form, an RR-MSIS instance over S_q gives a public row $\mathbf{a} = (a_1, \dots, a_m) \in S_q^m$ and asks for a non-zero short vector $\mathbf{x} = (x_1, \dots, x_m) \in S_q^m$ such that

$$\sum_{i=1}^m a_i x_i = 0 \pmod{q}.$$

Shortness is measured coefficient-wise in the signed representative of S_q . The relations extracted from Fiat-Shamir forgeries have the same structure, but with the public row and the target

specialized to the verification equation of **Octarine**. In the EUF-CMA proof the target is self-targeted: the adversary's $\mathbf{w}_1$ fixes the right-hand side and must be consistent with the random oracle through $\tilde{c} = H(\mu \parallel \mathbf{w}_1)$ and $c = \text{SampleInBall}(\tilde{c})$. In the SUF-CMA proof, subtracting two accepting signatures with the same recovered $\mathbf{w}_1$ cancels this target and leaves a homogeneous RR-MSIS relation.

The following two bounds are the ones used for the unforgeability estimates in Table 2:

$$B_{\text{EUF}} = \max \left\{ \gamma_1 - \beta, 2\gamma_2 + 1 + \frac{q}{p}\tau 2^{d-1} \right\},$$

and

$$B_{\text{SUF}} = \max \{ 2(\gamma_1 - \beta), 4\gamma_2 + 2 \}.$$

The EUF-CMA extraction gives an inhomogeneous, self-targeted RR-MSIS relation: given $(\mathbf{A}, \mathbf{b})$, find $(\mu, \mathbf{z}, \tilde{c}, c, \mathbf{r}, \mathbf{w}_1)$, where $\mathbf{z}, \mathbf{r} \in S_q$, $\tilde{c} \in \{0, 1\}^{|\tilde{c}|}$, and $c \in \mathcal{C}_\tau$, such that

$$c = \text{SampleInBall}(\tilde{c}), \quad \tilde{c} = H(\mu \parallel \mathbf{w}_1),$$

with $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\|\mathbf{r}\|_\infty \leq B_{\text{EUF}}$ such that

$$\mathbf{A}\mathbf{z} - c\frac{q}{p}\mathbf{b} - \mathbf{r} = 2\gamma_2\mathbf{w}_1 \pmod{q}.$$

The short unknowns have $2kn + n$ coefficient positions, namely kn for $\mathbf{z}$, kn for $\mathbf{r}$, and n for the constant-component challenge. In Table 2 we conservatively estimate the cost after embedding the challenge into a full S_q element, which gives the larger dimension $3kn$. With this convention, the SUF-CMA extraction gives the homogeneous RR-MSIS relation of dimension $2kn$: given $\mathbf{A}$, find a non-zero pair $(\Delta\mathbf{z}, \Delta\mathbf{r}) \in S_q^2$ with

$$\|\Delta\mathbf{z}\|_\infty < 2(\gamma_1 - \beta), \quad \|\Delta\mathbf{r}\|_\infty \leq 4\gamma_2 + 2, \quad \mathbf{A}\Delta\mathbf{z} - \Delta\mathbf{r} = 0 \pmod{q}.$$

Both problems are the radical-ring analogues of the MSIS problems used in the security analysis of Fiat-Shamir with aborts signatures. The reductions below are stated in the classical random oracle model. A QROM proof would require the additional Fiat-Shamir reprogramming analysis used in QROM treatments of Fiat-Shamir with aborts signatures; we do not claim such a proof here.

The reduction also separates the purely computational losses from the statistical losses of the signing simulator. Let Δ_{sim} denote the maximum statistical distance, over the public keys in the post-RR-LWR hybrid and over signing-query messages, between the distribution of an accepted real signing transcript and the transcript output by the simulator. This term includes the rejection-sampling conditions, the distribution of the low bits and high bits of $\mathbf{A}\mathbf{y}$ for bounded $\mathbf{y}$, and the canonical hint distribution. Let $2^{-\alpha}$ denote an upper bound on the maximum point probability of the simulated commitment high bits $\mathbf{w}_1$. These quantities are not discharged by the RR-LWR or RR-MSIS assumptions themselves; for a concrete parameter set they must be bounded separately, or treated explicitly as statistical assumptions in the concrete security estimate.

Theorem 4 (EUF-CMA security). *Assume that the key-generation **RR-LWR** problem of Definition 4 is hard for the secret distribution D_s , that the self-targeted RR-MSIS relation above is hard for bound B_{EUF} , that H is modeled as a random oracle, and that the signing simulator satisfies the statistical bounds Δ_{sim} and $2^{-\alpha}$ described above. Then every adversary making at most Q_H random-oracle queries and Q_S signing queries has advantage bounded by*

$$\begin{aligned} & \text{Adv}_{\text{kg}, q, p, D_s}^{\text{d-rr-lwr}}(\mathcal{B}_1) + \Delta_A + \text{Adv}_{q, B_{\text{EUF}}}^{\text{st-rr-msis}}(\mathcal{B}_2) + (Q_H + 1)2^{-|\tilde{c}|} + (Q_H + Q_S + 1)2^{-|\mu|} \\ & + Q_S(Q_H + Q_S)2^{-\alpha} + Q_S\Delta_{\text{sim}}, \end{aligned}$$

where $\Delta_{\mathbf{A}}$ is the distinguishing advantage for replacing the seeded public-matrix sampler by an explicit uniform matrix, and where the point probability bound means that, for every fixed digest μ and every fixed value $\mathbf{u}$, the sampled $\mathbf{w}_1$ satisfies $\Pr[\mathbf{w}_1 = \mathbf{u}] \leq 2^{-\alpha}$. The term $(Q_H + Q_S + 1)^2 2^{-|\mu|}$ accounts only for collisions among distinct inputs to the first-stage message hash $M \mapsto H(tr \parallel M)$.

Proof. The public key contains $(\rho, \mathbf{b}_1)$, where ρ expands to $\mathbf{A}$ and $\mathbf{b}_1$ is obtained from $\mathbf{b} = \lfloor (p/q)\mathbf{A}\mathbf{s}_1 \rfloor$. First replace the seeded matrix $D_{\mathbf{A}}(\mathbf{S}_{q,n,k}; \rho)$ by an explicit uniform $\mathbf{A} \leftarrow S_q$, at cost $\Delta_{\mathbf{A}}$. The key-generation RR-LWR hop is applied to the full challenge $(\mathbf{A}, \mathbf{b}) \in S_q \times S_p$: the reduction computes $(\mathbf{b}_1, \mathbf{b}_0) = \text{Power2Round}_p(\mathbf{b}, 2^d)$, gives only the expanded public key $(\mathbf{A}, \mathbf{b}_1)$ to the adversary, and keeps $\mathbf{b}_0$ for the later extraction step. By the key-generation RR-LWR assumption, replacing $\mathbf{b}$ by a uniform element of S_p changes the adversary's advantage by at most $\text{Adv}_{\text{kg},q,p,D_s}^{\text{d-rr-lwr}}$. After this hop the reduction need not know a secret key. It answers signing queries with the standard Fiat-Shamir with aborts transcript simulator. On a query M_i , it computes $\mu_i = H(tr \parallel M_i)$. If the same message is queried again, the simulator reuses the same μ_i , as the real random oracle would. The bad message-digest event is only that two distinct inputs $tr \parallel M \neq tr \parallel M'$ receive the same digest. For the current query, the simulator samples an accepted verification transcript $(\tilde{c}_i, \mathbf{z}_i, \mathbf{h}_i, \mathbf{w}_{1,i})$ according to the public transcript distribution and programs $H(\mu_i \parallel \mathbf{w}_{1,i}) = \tilde{c}_i$. It declares a bad event and resamples the transcript if the point $(\mu_i, \mathbf{w}_{1,i})$ was already fixed by a random-oracle query or by a previous simulated signature, including an earlier signature on the same message. These bad events contribute the $(Q_H + Q_S + 1)^2 2^{-|\mu|}$ and $Q_S(Q_H + Q_S)2^{-\alpha}$ terms in the theorem statement. Conditioned on avoiding them, the transcript distribution differs from real signatures by at most Δ_{sim} per signing query. In particular, the accepted response hides $\mathbf{s}_1$ by the usual rejection sampling argument: for each coefficient, the condition $\|\mathbf{z}\|_{\infty} < \gamma_1 - \beta$ selects an interval of the same length for every possible coefficient of $\mathbf{c}\mathbf{s}_1$. The analogous low-bit and hint distributions depend on the distribution of $\mathbf{A}\mathbf{y}$ when $\mathbf{y}$ is sampled from the bounded signing distribution rather than uniformly modulo q ; any deviation between the real and simulated low-bit, high-bit, and hint distributions is accounted for in Δ_{sim} .

Let (M^*, σ^*) be a valid forgery in the simulated experiment, and write

$$\sigma^* = \text{sigEncode}(\tilde{c}^*, \mathbf{z}^*, \mathbf{h}^*), \quad c^* = \text{SampleInBall}(\tilde{c}^*), \quad \mu^* = H(tr \parallel M^*).$$

Verification computes

$$\mathbf{v}^* = \mathbf{A}\mathbf{z}^* - c^* \frac{q}{p} \mathbf{b}_1 2^d$$

and recovers a value $\mathbf{w}_1^*$ from $\text{UseHint}_q(\mathbf{h}^*, \mathbf{v}^*, 2\gamma_2)$. Since the signature verifies, the random oracle value at $(\mu^*, \mathbf{w}_1^*)$ is $\tilde{c}^*$. If this oracle input was never queried before the forgery was output, the adversary guessed an independent $|\tilde{c}|$ -bit random-oracle value. The theorem uses the conservative bound $(Q_H + 1)2^{-|\tilde{c}|}$ for this final challenge-guessing event.

It remains to handle the case where the query $(\mu^*, \mathbf{w}_1^*)$ appears in the random-oracle transcript. From the definition of UseHint_q , for every accepting signature there is an element $\mathbf{e}_h^* \in S_q$ with $\|\mathbf{e}_h^*\|_{\infty} \leq 2\gamma_2 + 1$ such that

$$\mathbf{A}\mathbf{z}^* - c^* \frac{q}{p} \mathbf{b}_1 2^d = 2\gamma_2 \mathbf{w}_1^* + \mathbf{e}_h^* \pmod{q}.$$

Using $\mathbf{b} = \mathbf{b}_1 2^d + \mathbf{b}_0$, this becomes

$$\mathbf{A}\mathbf{z}^* - c^* \frac{q}{p} \mathbf{b} = 2\gamma_2 \mathbf{w}_1^* + \mathbf{r}^* \pmod{q},$$

where

$$\mathbf{r}^* = \mathbf{e}_h^* - c^* \frac{q}{p} \mathbf{b}_0.$$

Because c^* has Hamming weight τ and every coefficient of $\mathbf{b}_0$ lies in $[-2^{d-1}, 2^{d-1} - 1]$, we have

$$\left\| c^* \frac{q}{p} \mathbf{b}_0 \right\|_\infty \leq \frac{q}{p} \tau 2^{d-1}.$$

Therefore

$$\|\mathbf{r}^*\|_\infty \leq 2\gamma_2 + 1 + \frac{q}{p} \tau 2^{d-1}, \quad \|\mathbf{z}^*\|_\infty < \gamma_1 - \beta.$$

The tuple $(\mu^*, \mathbf{z}^*, \tilde{c}^*, c^*, \mathbf{r}^*, \mathbf{w}_1^*)$ is thus a valid solution to the self-targeted RR-MSIS relation because $\tilde{c}^* = \mathcal{H}(\mu^* \parallel \mathbf{w}_1^*)$ and $c^* = \text{SampleInBall}(\tilde{c}^*)$. Since $c^* \in \mathcal{C}_\tau$ is non-zero, the extracted relation is non-trivial. Any EUF-CMA adversary with non-negligible success therefore gives a solver for this self-targeted RR-MSIS problem, except with the random-oracle guessing, message-digest collision, programming, and simulation losses already stated. $\square$

Theorem 5 (SUF-CMA security). *Under the assumptions of Theorem 4, and assuming in addition that the homogeneous RR-MSIS relation above is hard for bound B_{SUF} , Octarine is SUF-CMA secure. More precisely, the bound of Theorem 4 is augmented by*

$$\text{Adv}_{q, B_{\text{SUF}}}^{\text{rr-msis}}(\mathcal{B}_3),$$

for the homogeneous RR-MSIS problem with bound B_{SUF} .

Proof. Every SUF-CMA forgery on a message that was never signed is already an EUF-CMA forgery, so Theorem 4 applies. It remains to consider a fresh valid signature on a message M for which the signing oracle previously returned one or more valid signatures $\sigma^{(j)} = \text{sigEncode}(\tilde{c}^{(j)}, \mathbf{z}^{(j)}, \mathbf{h}^{(j)})$. Let the forged signature be

$$\sigma^* = \text{sigEncode}(\tilde{c}^*, \mathbf{z}^*, \mathbf{h}^*).$$

Let $\mu = \mathcal{H}(tr \parallel M)$, and let $\mathbf{w}_1^*$ be the high bits recovered by verifying σ^* . If $\mathbf{w}_1^*$ is different from every recovered high-bit value $\mathbf{w}_1^{(j)}$ of the previously returned signatures for M , then $(\mu, \mathbf{w}_1^*)$ is fresh with respect to the programmed signing points. The proof of Theorem 4 therefore gives either a self-targeted RR-MSIS solution or a random-oracle guessing event.

We may therefore restrict attention to the case where $\mathbf{w}_1^* = \mathbf{w}_1^{(j)}$ for some previously returned signature $\sigma^{(j)}$. Since the random oracle is consistent,

$$\tilde{c}^* = \mathcal{H}(\mu \parallel \mathbf{w}_1^*) = \mathcal{H}(\mu \parallel \mathbf{w}_1^{(j)}) = \tilde{c}^{(j)}.$$

Thus also $c^* = c^{(j)} = \text{SampleInBall}(\tilde{c}^*)$.

For $\sigma^{(j)}$ and the forged signature, the verification equation gives elements $\mathbf{e}_h^{(j)}, \mathbf{e}_h^* \in S_q$ satisfying

$$\mathbf{A} \mathbf{z}^{(j)} - c^* \frac{q}{p} \mathbf{b}_1 2^d = 2\gamma_2 \mathbf{w}_1^* + \mathbf{e}_h^{(j)} \pmod{q}$$

and

$$\mathbf{A} \mathbf{z}^* - c^* \frac{q}{p} \mathbf{b}_1 2^d = 2\gamma_2 \mathbf{w}_1^* + \mathbf{e}_h^* \pmod{q},$$

with $\|\mathbf{e}_h^{(j)}\|_\infty, \|\mathbf{e}_h^*\|_\infty \leq 2\gamma_2 + 1$. Subtracting these two equations cancels the public-key term and the target:

$$\mathbf{A}(\mathbf{z}^* - \mathbf{z}^{(j)}) - (\mathbf{e}_h^* - \mathbf{e}_h^{(j)}) = 0 \pmod{q}.$$

Moreover,

$$\|\mathbf{z}^* - \mathbf{z}^{(j)}\|_\infty < 2(\gamma_1 - \beta), \quad \|\mathbf{e}_h^* - \mathbf{e}_h^{(j)}\|_\infty \leq 4\gamma_2 + 2.$$

Thus

$$(\Delta \mathbf{z}, \Delta \mathbf{r}) = (\mathbf{z}^* - \mathbf{z}^{(j)}, \mathbf{e}_h^* - \mathbf{e}_h^{(j)})$$

is a solution to the homogeneous RR-MSIS relation with bound B_{SUF} .

It remains only to see that the solution is non-zero. If $\Delta \mathbf{z} = 0$, then the same value $\mathbf{v} = \mathbf{A}\mathbf{z}^{(j)} - c^*(q/p)\mathbf{b}_1 2^d$ is used in both verifications. For fixed $\mathbf{v}$ and fixed recovered high bits $\mathbf{w}_1^*$, the hint vector accepted by UseHint_q is determined coefficient-wise. The signature encoding is canonical: unused hint indices must be zero, indices inside each polynomial are strictly increasing, and the decoder rejects non-canonical encodings. Hence the same $\tilde{c}^*$, $\mathbf{z}^* = \mathbf{z}^{(j)}$, and $\mathbf{w}_1^*$ force the same $\mathbf{h}$ and therefore the same signature bytes. This contradicts freshness in the SUF-CMA experiment. Consequently the extracted RR-MSIS solution is non-zero. $\square$

6.1.3 Beyond-unforgeability properties

The challenge seed carried in an Octarine signature also gives the binding mechanism used in the BUFF analysis of Fiat–Shamir signatures [CDF⁺20]. BUFF security refers to properties beyond ordinary unforgeability: malicious-strong universal exclusive ownership (M-S-UEO), where one signature should not verify under two adversarially chosen public keys possibly for two different messages; message-bound signatures (MBS), where one signature under a fixed public key should not verify two different messages; and non-resignability properties, where a signature on an unknown message should not allow producing a related signature under a different public key for that same unknown message.

Subsequent work showed that the original strong NR formulation and the corresponding proof strategy for the BUFF transform are too strong in general, and introduced a weakened achievable formulation [DFHS24]. The currently available positive result for the BUFF structure proves weak-NR in an idealized model for high-min-entropy message distributions [DFH⁺24]. The statement below therefore claims the ordinary BUFF ownership properties in the random-oracle model, and only the corresponding weak-NR property in that idealized high-min-entropy setting. The explicit canonical-decoding condition is important in the malicious-key setting: weak or malformed public keys are precisely where BUFF ownership properties can fail [DS24].

Let

$$H_{\text{BUFF}}(pk, M, \mathbf{w}_1) = H_{\text{ch}}(H_{\text{mu}}(H_{\text{tr}}(pk; |tr|), M; |\mu|), \mathbf{w}_1; |\tilde{c}|)$$

denote the composed challenge hash, with the fixed output length used for $\tilde{c}$. Each of H_{tr} , H_{mu} , and H_{ch} uses the domain-separated, length-delimited encoding specified in Section 4. The actual algorithms compute $tr = H_{\text{tr}}(pk; |tr|)$, $\mu = H_{\text{mu}}(tr, M; |\mu|)$, and $\tilde{c} = H_{\text{BUFF}}(pk, M, \mathbf{w}_1)$; the value $\tilde{c}$ is included in the signature and verification explicitly checks that the same digest is obtained from the recovered $\mathbf{w}_1$. Thus the BUFF digest is already present in the Fiat–Shamir challenge seed, and no generic transformation or extra signature component is needed.

Theorem 6 (BUFF properties). *Assume that public-key decoding accepts only canonical byte encodings and that the domain-separated, length-delimited encoding of $(pk, M, \mathbf{w}_1)$ is injective for every accepted public-key byte string. In the random oracle model, every adversary making at most Q_H random-oracle queries has M-S-UEO and MBS advantage bounded by*

$$(Q_H + 6)^2 \left(2^{-|tr|} + 2^{-|\mu|} + 2^{-|\tilde{c}|} \right),$$

which is at most $3(Q_H + 6)^2 2^{-|\tilde{c}|}$ when the intermediate hash outputs are at least $|\tilde{c}|$ bits. In particular, the exclusive-ownership family covered by M-S-UEO also covers the S-CEO and S-DEO notions of [CDF⁺20]. In the idealized weak-NR model of [DFH⁺24], for high-min-entropy message distributions, the same challenge binding gives Octarine the weak-NR property established for the BUFF structure. We do not claim the original strong NR notion of [CDF⁺20].

Proof. We view the hash calls in H_{BUFF} as random-oracle calls. Collision resistance of the composed challenge hash is therefore the corresponding random-oracle collision event; when two composed outputs are equal, the collision can be extracted by walking the chain through H_{tr} , H_{mu} , and the final H_{ch} call.

Consider first an adversary that outputs a single signature $\sigma = \text{sigEncode}(\tilde{c}, \mathbf{z}, \mathbf{h})$ that verifies for two tuples (pk_1, M_1) and (pk_2, M_2) , where the public keys may be maliciously chosen. Verification first decodes the public-key bytes; by assumption, non-canonical encodings are rejected and every accepted public key has a unique byte representation. It then parses the same $\tilde{c}$, recomputes the challenge $c = \text{SampleInBall}(\tilde{c})$, recovers high bits $\mathbf{w}_{1,1}$ and $\mathbf{w}_{1,2}$ from the two public-key views, and accepts only if

$$\tilde{c} = H_{\text{BUFF}}(pk_1, M_1, \mathbf{w}_{1,1}) = H_{\text{BUFF}}(pk_2, M_2, \mathbf{w}_{1,2}).$$

For M-S-UEO the experiment has $pk_1 \neq pk_2$ and allows $M_1 \neq M_2$; for MBS it has $pk_1 = pk_2$ and $M_1 \neq M_2$. With injective encodings of all accepted triples $(pk, M, \mathbf{w}_1)$, either winning event therefore gives a collision of the underlying random oracle somewhere in the composed hash chain, except with negligible probability. The same M-S-UEO statement implies the weaker exclusive-ownership variants S-CEO and S-DEO. The $Q_H + 6$ term in the theorem counts the adversary’s oracle queries together with the at most six random-oracle evaluations needed to recompute the two three-step challenge-hash chains in the winning check.

For non-resignability, the above collision argument is not by itself a proof of the original strong NR notion: that notion is known to be unachievable in the generic BUFF setting, and the original proof path has a gap [DFHS24]. Instead we use the weaker formulation proved for the BUFF structure in the idealized hide-and-seek model [DFH⁺24]. The **Octarine** signature already contains the BUFF challenge digest

$$\tilde{c} = H_{\text{BUFF}}(pk, M, \mathbf{w}_1),$$

and verification recomputes it from the public key, the message, and the recovered high bits. In the hide-and-seek view, the base signature component is the native Fiat–Shamir response-and-hint data $(\mathbf{z}, \mathbf{h})$ together with the deterministic verifier reconstruction of $\mathbf{w}_1$; the appended BUFF digest is the challenge seed $\tilde{c}$, which **Octarine** already stores rather than adding as a second signature component. Thus the inputs exposed to the idealized weak-NR game are the BUFF tuple $(pk, M, \mathbf{w}_1)$ with digest $\tilde{c}$. For messages drawn from the required high-min-entropy distribution, a successful re-signing attack therefore gives either message prediction, an underlying random-oracle collision, or a break of the weak-NR game analyzed in [DFH⁺24]. $\square$

6.2 Practical Security

6.2.1 Physical Security

The signing algorithm uses the long-term signing key and is therefore the primary target for side-channel attacks. Given the similarity in the high-level Fiat–Shamir-with-aborts structure, such attacks are expected to resemble those studied for ML-DSA [RCDB24]. At the same time, several design choices in **Octarine** are intended to simplify hardened implementations. In particular, the use of power-of-two moduli makes sampling, rounding, decomposition, and bit-selection operations naturally compatible with fixed-time implementations based on shifts, masks, and additions. These operations are also well suited to masking-based countermeasures, as they avoid the additional complexity typically introduced by arithmetic modulo an odd prime.

We therefore expect the overhead of countermeasures such as masking to be lower for **Octarine** than for ML-DSA. This expectation is consistent with previous work on masking Dilithium, which observed that replacing the prime modulus by a power-of-two modulus can substantially reduce masking overhead [MGTF19]. A detailed side-channel evaluation of **Octarine** is left as future work, but its power-of-two arithmetic and regular sampling procedures provide a favorable starting point for constant-time and masked implementations.

7 Performance Evaluation

7.1 Arithmetic Implementations

The ARCANÉ suite deliberately does not prescribe a single polynomial-multiplication strategy for either the KEM (Mithril) or the signature scheme (Octarine). Polynomial products may therefore be implemented using schoolbook multiplication, Karatsuba, Toom–Cook, NTT-based methods, Kronecker-style techniques, or any other approach that is suitable for the target platform. This is an implementation choice rather than part of the specification, and gives implementers the flexibility to select the arithmetic algorithm that best matches the available hardware, performance objectives, memory constraints, and hardening requirements.

Both schemes use power-of-two moduli for their specified arithmetic. This makes reduction modulo the scheme modulus particularly simple and keeps rounding, packing, and decomposition operations close to bit extraction. Moreover, the Octarine modulus has roughly twice the bit length of the Mithril modulus. This relation is useful for shared software and hardware implementations: a datapath dimensioned for the larger Octarine coefficients can naturally accommodate Mithril arithmetic as well, while smaller subroutines can often be reused inside the larger-degree Octarine computations.

Although the specified moduli are powers of two, implementations may still use auxiliary NTT primes internally. For example, a product in $\mathbb{Z}_{q_K}[x]/(x^n + 1)$ can be computed by lifting the inputs to signed integers, mapping them to an auxiliary prime field $\mathbb{Z}_\pi$, performing a negacyclic NTT multiplication modulo π , and then reducing the recovered integer result modulo q_K . The auxiliary prime π should support the required negacyclic transform length; in the usual setting this means that π contains a primitive $2n$ -th root of unity, equivalently $\pi \equiv 1 \pmod{2n}$. It must also be large enough that the relevant integer convolution coefficients do not wrap around modulo π . For Mithril, where the scheme modulus is in the 12–16 bit range, a single 20–31 bit NTT prime is typically sufficient.

The same idea applies to several small products in Octarine. The challenge polynomial c is a sparse element of the base ring and is embedded into the constant component of the radical ring; multiplication by c therefore acts component-wise on the k base-ring components. Products such as cs_1 , cs_2 , and cb_0 have small coefficient growth because c has Hamming weight τ and the coefficients of s_1 , s_2 , and b_0 are bounded. Consequently, these products can often be evaluated using the same auxiliary-prime NTT infrastructure as the KEM arithmetic, provided that the chosen prime supports the Octarine base degree. If the KEM prime was selected only for the smaller Mithril transform length, an implementation can instead choose a compatible shared prime for the largest required transform length, or use a separate smaller auxiliary prime for these challenge products.

The larger Octarine products, in particular As_1 , Ay , and Az , require more care. These are products in the radical ring

$$S_{q_D, n, k} = (\mathbb{Z}_{q_D}[y]/(y^n + 1))[x]/(x^k - y - 2),$$

and are implemented as structured combinations of negacyclic base-ring multiplications. Since $q_D = 2^{24}$, the coefficients of y and z are roughly 20 bits, and $n = 1024$, a single 32-bit auxiliary prime does not in general provide enough dynamic range to recover the integer product coefficients without wraparound. A natural solution is therefore to use two auxiliary NTT primes π_1 and π_2 . The inputs are lifted to signed integers, reduced modulo each π_i , multiplied independently in

$$\mathbb{Z}_{\pi_i}[y]/(y^n + 1),$$

and the results are recombined by CRT modulo $M = \pi_1\pi_2$. If M is larger than twice the relevant signed coefficient-growth bound, the CRT result uniquely recovers the integer product coefficient, after which it is reduced modulo q_D .

For the Octarine parameter sets, the intended coefficient-growth bounds suggest that a product of two sub-32-bit NTT primes is sufficient: a modulus product in the rough range of 56–60 bits covers the dominant Ay and Az products, including the accumulation over the radical-ring components. This also makes it possible to choose one of the two primes to coincide with the auxiliary prime used for Mithril, provided that this prime supports the largest transform length required by the shared implementation. In this way, the KEM arithmetic can be reused directly as one branch of the two-prime CRT computation for Octarine.

Concretely, for each auxiliary prime π_i , an implementation computes the required base-ring products in the NTT domain modulo π_i . Additions, subtractions, component products, and the structured accumulations induced by the relation $x^k = y + 2$ are all performed independently modulo π_1 and modulo π_2 . The inverse CRT recombination need not be applied after each individual base multiplication. Instead, the implementation may keep all intermediate accumulations in the two-prime representation and recombine only after the full radical-ring product, or even after an entire matrix product, has been completed. This amortizes the CRT cost, which is particularly beneficial for Octarine because the products by the public object A dominate the arithmetic cost as the radical extension degree k grows.

Finally, the two auxiliary-prime branches can share essentially the same software structure. In a Montgomery implementation, the arithmetic layer can be parameterized by the prime π_i , the corresponding Montgomery constant $-\pi_i^{-1} \bmod R$, and the precomputed NTT roots of unity. The control flow, transform structure, pointwise multiplication routines, and accumulation logic remain otherwise unchanged. Thus, the same implementation framework can support both auxiliary primes with only minor parameter changes, reinforcing the overall objective of arithmetic reuse across the ARCANÉ family.

7.2 Performance Analyses

The cost profile of **Octarine** is determined by the fixed base ring, the radical extension degree, and the rejection loop in signing. All three parameter sets use $n = 1024$ and $q = 2^{24}$, while the radical extension degree is $k = 1, 2, 5$ for **Octarine-128**, **Octarine-256**, and **Octarine-512**, respectively. The expected numbers of Fiat–Shamir-with-aborts iterations are approximately 2.1, 2.3, and 3.6. Thus the relevant performance quantities are the amount of pseudorandom material expanded into ring coefficients, the cost of products by the public object $\mathbf{A}$, and the cost of the subsequent challenge products.

Sampling and expansion. The principal non-arithmetic costs come from expanding pseudorandom output into coefficient representations. The public object $\mathbf{A}$ is deterministically generated from the seed ρ and, in the radical-ring representation, consists of k polynomials with n coefficients modulo q . Since $q = 2^{24}$, public coefficients are obtained by fixed-width decoding of the output stream rather than by rejection sampling. The masking element $\mathbf{y}$ is sampled in each rejection-loop iteration from $[-\gamma_1, \gamma_1 - 1]$, so its expansion cost is multiplied by the expected number of signing attempts.

For $\mathbf{A}$ and $\mathbf{y}$, the sampling specification divides each polynomial into four domain-separated streams, each corresponding to a block of $n/4$ coefficients. This gives four independent extendable-output-function evaluations and four independent coefficient-decoding streams per polynomial; it is a regular, domain-separated construction, not a data-dependent sampling procedure. In contrast, $\mathbf{s}_1$ is sampled only once during key generation and uses only $\log_2(2\eta)$ bits per coefficient. The stored rounding-error component $\mathbf{s}_2$ is not sampled independently: it is derived from $\mathbf{A}\mathbf{s}_1$ and the rounded public value during key generation.

The challenge polynomial c generated by `SampleInBall` has Hamming weight τ . The measured implementations nevertheless use regular NTT-domain multiplication for the following products by c , rather than a multiplication rule whose memory accesses or control flow depend on the sparse support of c .

Products by the public object. The main arithmetic operations in the three algorithms are

$$\mathbf{A}\mathbf{s}_1, \quad \mathbf{A}\mathbf{y}, \quad \mathbf{A}\mathbf{z}.$$

These products are evaluated over signed integer lifts and two auxiliary NTT prime fields. For $\mathbf{u} \in \{\mathbf{s}_1, \mathbf{y}, \mathbf{z}\}$, the computation can be viewed abstractly as

$$\mathbf{A}\mathbf{u} = \text{red}_q \left(\text{CRT}_{\pi_1, \pi_2} \left(\text{INTT}_{\pi_1} (\hat{\mathbf{A}}^{(1)} \odot \hat{\mathbf{u}}^{(1)}), \right. \right. \\ \left. \left. \text{INTT}_{\pi_2} (\hat{\mathbf{A}}^{(2)} \odot \hat{\mathbf{u}}^{(2)}) \right) \right), \quad (8)$$

where the superscript indicates reduction to the corresponding auxiliary prime field, the hats denote NTT-domain representations, and red_q denotes reduction back to the signed representative modulo q . This is valid when $\pi_1\pi_2$ covers the relevant integer coefficient-growth bound. Here $\odot$ denotes the structured NTT-domain product and accumulation induced by multiplication in the radical ring, rather than a single component-wise product.

In a direct view, multiplication in the radical extension produces k output components and involves a structured combination of base-polynomial products. Optimized implementations may exploit the reduction relation $x^k - y - 2$ and reuse prepared representations, so the following expression should be read as an explanatory model rather than an exact cycle formula:

$$C_{\mathbf{A}}(k) \approx 2(kC_{\text{NTT}} + M(k)C_{\text{pt}} + kC_{\text{INTT}}) + kC_{\text{CRT}}, \quad (9)$$

where $M(k) = O(k^2)$ in the direct radical-ring view. The model separates the two auxiliary prime fields, the transforms of the k components, the component products and accumulations, and the final CRT recovery. It explains why products by $\mathbf{A}$ are expected to be the principal arithmetic operations as k increases, while leaving exact costs to the measured implementation results.

Challenge products. Products by the challenge polynomial have a different structure. The challenge c is a single base-ring polynomial embedded into the constant component of the radical ring, so multiplication by c acts component-wise on

$$\mathbf{s}_1, \quad \mathbf{s}_2, \quad \mathbf{b}_0, \quad \mathbf{b}_1.$$

In the measured implementations, c is transformed once and then used in regular component-wise NTT products. After this transform, the cost of a challenge-product group is therefore roughly linear in k , and it avoids the cross-component public-product structure present in products by $\mathbf{A}$.

Algorithm-level decomposition. At the level of the three main algorithms, this gives the following coarse decomposition:

$$C_{\text{KeyGen}} \approx C_{\text{seed}} + C_{D_{\mathbf{A}}} + C_{D_{\mathbf{s}_1}} + C_{\mathbf{A}}(k) + O(kn), \quad (10)$$

$$C_{\text{Sign}} \approx C_{\text{pre}} + r_{\text{FS}} \left(C_{D_{\mathbf{y}}} + C_{\mathbf{A}}(k) + C_{w_1} + C_{\text{SampleInBall}} + C_{c\text{-prod}} \right. \\ \left. + C_{\text{MakeHint}} \right) + C_{\text{pack}}, \quad (11)$$

$$C_{\text{Verify}} \approx C_{\text{unpack}} + C_{D_{\mathbf{A}}} + C_{\mathbf{A}}(k) + C_{\text{SampleInBall}} + C_{c\mathbf{b}_1, \text{hint}} + C_{\text{hash}}. \quad (12)$$

Here r_{FS} denotes the expected number of rejection-loop iterations. The terms multiplied by r_{FS} are the per-attempt operations in the rejection loop, including operations that are reached only after the earlier rejection tests pass; the final signature encoding cost C_{pack} is incurred once after an accepting iteration. The term $C_{c\text{-prod}}$ collects the challenge products with $\mathbf{s}_2$, $\mathbf{s}_1$, and $\mathbf{b}_0$ in the signing loop. The term $C_{c\mathbf{b}_1, \text{hint}}$ includes the challenge product with $\mathbf{b}_1$, the corresponding high-bit reconstruction, and the hint application in verification. The terms $C_{D_{\mathbf{A}}}$ and $C_{D_{\mathbf{y}}}$ include the

cost of expanding pseudorandom output into ring coefficients. The implementation results in the next subsection test this decomposition by separating whole-function timings from instrumented stage timings.

7.3 Performance Test

Experimental setup. All software measurements in this section were obtained on one core of an Intel Xeon E5-2686 v4 processor reported at 2.30 GHz by `/proc/cpuinfo`. The machine has 3.8 GiB of RAM and ran Ubuntu GNU/Linux with Linux kernel 7.0.0-1006-aws on x86_64. The compiler was gcc 15.2.0. Benchmarks were run serially, with each benchmark process pinned to a single CPU. Each individual run reports the median of 1000 measured executions, and the unit reported by the harness is cycles/ticks.

The build configurations were as follows. The reference implementation using provided pseudoXOF was compiled at `-O3` with the reference Makefile warning flags; see the table below. The SHA3/SHAKE AVX2 implementation uses the same AVX2 arithmetic flags as the provided-pseudoXOF AVX2 implementation, with the SHA3/SHAKE backend and four-way Keccak support.

Implementation	Compiler flags
Reference pseudoXOF	<code>-O3</code> , plus reference Makefile warning flags
AVX2 pseudoXOF	<code>-O3</code> , <code>-mavx2</code> , <code>-mbmi2</code> , <code>-flto</code> <code>-fno-semantic-interposition</code> , <code>-funroll-loops</code> <code>-fno-inline-functions-called-once</code> , plus warning flags
ML-DSA AVX2	<code>-O3</code> , <code>-mavx2</code> , <code>-mpopcnt</code> <code>-march=native</code> , <code>-mtune=native</code>

The ML-DSA comparison implementation was the official CRYSTALS-Dilithium AVX2 implementation, checked out at git commit

6e00625c5b29f516c6de973fe2ee2fbb150973f9.

Performance of reference and AVX2 implementation with pseudoXOF. Table 3 first reports the whole-function cost of the submitted implementations. The two implementations use the same pseudoXOF interface, so this comparison isolates the contribution of the AVX2 arithmetic and packing layers under the submitted hash interface.

Table 3: Whole-function performance of the reference and AVX2 implementations using pseudoXOF. Values are median cycles/ticks; speedup is Reference/AVX2.

Parameter set	Operation	Reference	AVX2	Speedup
Octarine-128	KeyGen	896 744	496 887	1.80×
	Sign	2 369 061	1 131 795	2.09×
	Verify	1 062 976	493 150	2.16×
Octarine-256	KeyGen	1 757 081	922 466	1.90×
	Sign	4 754 636	2 337 069	2.03×
	Verify	2 086 376	964 220	2.16×
Octarine-512	KeyGen	4 711 052	2 216 628	2.13×
	Sign	17 112 796	8 337 111	2.05×
	Verify	5 511 306	2 336 848	2.36×

Tables 4–6 give the corresponding instrumented breakdowns. The stage measurements are reported as medians for the same serial benchmark run and should be read as stage-level measurements rather than as an exact additive reconstruction of the whole-function medians.

Table 4: Octarine-128 using pseudoXOF breakdown. Values are median cycles/ticks; speedup is Reference/AVX2.

Stage	Reference	AVX2	Speedup
KeyGen total	900 817	498 804	1.81×
KeyGen random/hash	29 735	27 545	1.08×
KeyGen D_A and preparation	283 242	272 794	1.04×
KeyGen D_{s_1}	53 941	48 889	1.10×
KeyGen A_{s_1}	389 988	29 000	13.45×
KeyGen hash/store	125 275	115 794	1.08×
Sign total	2 367 737	1 135 682	2.08×
Sign prepare A	397 040	272 558	1.46×
Sign unpack/transform secret	170 301	17 947	9.49×
Sign hash μ, ρ''	64 508	59 844	1.08×
Sign D_y	659 362	613 577	1.07×
Sign transform y	225 517	24 109	9.35×
Sign A_y	332 398	33 370	9.96×
Sign w_1 /challenge hash	46 401	40 955	1.13×
Sign SampleInBall and transform	134 865	34 276	3.93×
Sign cs_2 and check	147 580	15 512	9.51×
Sign cs_1 and check	73 656	7 707	9.56×
Sign cb_0 and hint	74 574	8 538	8.73×
Sign pack signature	7 314	3 754	1.95×
Verify total	1 064 732	495 839	2.15×
Verify unpack/check	4 837	1 491	3.24×
Verify prepare A	283 158	273 427	1.04×
Verify A_z	388 774	28 892	13.46×
Verify SampleInBall and transform	67 795	17 318	3.91×
Verify unpack hint	844	635	1.33×
Verify cb_1 and high bits	131 036	14 665	8.94×
Verify hash compare	170 207	157 216	1.08×

Table 5: Octarine-256 using pseudoXOF breakdown. Values are median cycles/ticks; speedup is Reference/AVX2.

Stage	Reference	AVX2	Speedup
KeyGen total	1 798 146	920 670	1.95×
KeyGen random/hash	29 799	27 754	1.07×
KeyGen D_A and preparation	566 443	548 733	1.03×
KeyGen D_{s_1}	71 356	65 731	1.09×
KeyGen A_{s_1}	876 998	61 067	14.36×
KeyGen hash/store	216 116	199 375	1.08×
Sign total	4 875 423	2 340 256	2.08×
Sign prepare A	812 532	549 769	1.48×
Sign unpack/transform secret	356 823	36 009	9.91×
Sign hash μ, ρ''	64 937	60 251	1.08×
Sign D_y	1 413 432	1 301 403	1.09×
Sign transform y	473 736	50 484	9.38×
Sign A_y	816 980	71 473	11.43×
Sign w_1 /challenge hash	140 254	125 440	1.12×
Sign SampleInBall and transform	154 201	45 286	3.41×
Sign cs_2 and check	230 142	23 994	9.59×
Sign cs_1 and check	152 426	15 309	9.96×
Sign cb_0 and hint	154 134	16 892	9.12×
Sign pack signature	12 727	8 809	1.44×
Verify total	2 147 015	959 349	2.24×
Verify unpack/check	12 142	3 015	4.03×
Verify prepare A	566 007	549 382	1.03×
Verify A_z	877 855	60 561	14.50×
Verify SampleInBall and transform	77 550	22 792	3.40×
Verify unpack hint	2 049	1 531	1.34×
Verify cb_1 and high bits	276 043	30 060	9.18×
Verify hash compare	305 710	282 061	1.08×

Table 6: Octarine-512 using pseudoXOF breakdown. Values are median cycles/ticks; speedup is Reference/AVX2.

Stage	Reference	AVX2	Speedup
KeyGen total	4 697 799	2 223 798	2.11×
KeyGen random/hash	30 080	28 054	1.07×
KeyGen D_A and preparation	1 441 244	1 401 922	1.03×
KeyGen D_{s_1}	178 394	163 880	1.09×
KeyGen $\mathbf{As}_1$	2 541 761	169 657	14.98×
KeyGen hash/store	465 667	430 248	1.08×
Sign total	17 134 788	8 319 675	2.06×
Sign prepare $\mathbf{A}$	2 051 675	1 400 079	1.47×
Sign unpack/transform secret	898 403	91 148	9.86×
Sign hash μ, ρ''	64 779	60 492	1.07×
Sign D_y	5 773 898	5 361 874	1.08×
Sign transform y	1 794 654	194 953	9.21×
Sign $\mathbf{Ay}$	3 991 001	318 462	12.53×
Sign w_1 /challenge hash	589 284	532 365	1.11×
Sign SampleInBall and transform	302 456	134 714	2.25×
Sign cs_2 and check	790 328	79 662	9.92×
Sign cs_1 and check	399 837	38 966	10.26×
Sign cb_0 and hint	396 684	42 337	9.37×
Sign pack signature	30 870	18 845	1.64×
Verify total	5 496 793	2 343 033	2.35×
Verify unpack/check	14 960	8 124	1.84×
Verify prepare $\mathbf{A}$	1 441 628	1 403 678	1.03×
Verify $\mathbf{Az}$	2 531 417	169 927	14.90×
Verify SampleInBall and transform	101 761	45 093	2.26×
Verify unpack hint	4 393	3 162	1.39×
Verify cb_1 and high bits	701 161	75 916	9.24×
Verify hash compare	682 770	625 908	1.09×

The breakdown supports the decomposition from the previous subsection. The large products by $\mathbf{A}$, namely $\mathbf{As}_1$, $\mathbf{Ay}$, and $\mathbf{Az}$, obtain roughly one order of magnitude speedup in the optimized arithmetic. The same is true for the regular NTT-domain challenge products. In contrast, stages dominated by the pseudoXOF interface and final hash reconstruction improve only slightly. This explains why the whole-function speedups in Table 3 are about 2×, even though the dominant arithmetic stages are accelerated much more strongly.

Performance comparison using SHA3/SHAKE. Table 7 compares the SHA3/SHAKE-based AVX2 implementation of Octarine with the AVX2 implementation of ML-DSA. The comparison is made between parameter sets with nearby, rather than exactly matching, security levels: Octarine-128 is compared with ML-DSA-44, and Octarine-256 with ML-DSA-87. We omit Octarine-512 from this table, since it targets a substantially higher security level than any ML-DSA parameter set. Lower values indicate better performance.

Table 7: Octarine vs. ML-DSA using SHA3/SHAKE AVX2 implementations. Values are median cycles/ticks; bold marks the faster median.

Comparison	Operation	Octarine AVX2	ML-DSA AVX2	Faster median
Octarine-128 vs. ML-DSA-44	KeyGen	79 994	87 164	Octarine , 1.09×
	Sign	204 968	203 781	ML-DSA 0.994×
	Verify	102 704	91 568	ML-DSA 0.89×
Octarine-256 vs. ML-DSA-87	KeyGen	153 377	236 020	Octarine , 1.54×
	Sign	383 428	433 228	Octarine , 1.13×
	Verify	190 885	234 677	Octarine , 1.23×

Acknowledgments

We would like to thank the three doctoral students, Shicong Zhang, Chenxin Zhong, and Haochen Huang for their participation in the technical discussions on the algorithm design and for their contributions to the preparation of the submission materials. Their efforts supported different parts of the submission, with Shicong Zhang contributing to the algorithm description, Chenxin Zhong to the security proofs, and Haochen Huang to the software implementation.

A SageMath Script

A.1 Security estimates

We estimate the concrete hardness of the RR-LWR instance using the lattice-estimator framework of Albrecht et al. [APS15]. Specifically, we use the lattice estimator implementation from [malb/lattice-estimator](#), commit [be81b18](#).

The classical and quantum bit-security estimates as well as the refined attacks as reported in Table 2 can be reproduced by running the following script in the lattice-estimator repository. In order to obtain the security estimates for the 512-bit security level one needs to increase the upper bound on the blocksize by adjusting this value in `estimator/conf.py` to

```
1 max_beta = 3500

1 from estimator import *
2 from sage.all import binomial, log
3
4 q = 2**24
5 n = 1024
6 eta128 = 4
7 eta = 2
8
9 sig128 = {
10     "q": q,
11     "p": q / (2*eta128),
12     "n": n,
13     "k": 1,
14     "eta": eta128,
15     "gamma1": 2**18,
16     "gamma2": 2**17,
17     "tau": 16,
18     "d": 11,
19     "omega": 79,
20     "Xs": ND.UniformMod(2*eta128)
21 }
22
23 sig256 = {
24     "q": q,
25     "p": q/(2*eta),
26     "n": n,
27     "k": 2,
28     "eta": eta,
29     "gamma1": 2**19,
30     "gamma2": 2**18,
31     "tau": 36,
32     "d": 13,
33     "omega": 210,
34     "Xs": ND.UniformMod(2*eta)
```

```

35 }
36
37 eta = 2
38 sig512 = {
39     "q": q,
40     "p": q/(2*eta),
41     "n": n,
42     "k": 5,
43     "eta": eta,
44     "gamma1": 2**21,
45     "gamma2": 2**20,
46     "tau": 87,
47     "d": 14,
48     "omega": 399,
49     "Xs": ND.UniformMod(2*eta)
50 }
51
52 def evaluate_params(params):
53
54     q      = params["q"]
55     p      = params["p"]
56     n      = params["n"]
57     k      = params["k"]
58     gamma1 = params["gamma1"]
59     gamma2 = params["gamma2"]
60     tau    = params["tau"]
61     eta    = params["eta"]
62     d      = params["d"]
63     omega  = params["omega"]
64     beta   = tau*eta # Max size of challenge (in absolute value)
65     Xs     = params["Xs"]
66     Xe     = ND.UniformMod(2*q/(2*p))
67
68     assert(q/(2*p) == eta) # If this is not true, then c*s2 needs to be
69                             # handled separately
70
71
72     print('beta = ' + str(beta))
73
74     lwe = LWE.Parameters(
75         n=k*n,
76         q=q,
77         Xs=Xs,
78         Xe=Xe,
79         m=k*n,
80         tag="LWE"
81     )
82
83     LWE.estimate(lwe)
84
85     zeta = max((gamma1-beta), 2*gamma2 + 1 + q/p*2**(d-1)*tau)
86     print(zeta, gamma1-beta, 2*gamma2 + 1 + q/p*2**(d-1)*tau)
87     selftargetmsis = SIS.Parameters(
88         n=k*n,
89         q=q,
90         length_bound=zeta,
91         m=3*k*n,
92         norm=+Infinity,
93         tag="SIS"

```

```

92 )
93
94 print('selftargetmsis (forgery)')
95 print(selftargetmsis)
96 SIS.estimate(selftargetmsis)
97
98 zeta_ = max(2*(gamma1-beta), 4*gamma2 + 2)
99 print(zeta_, 2*(gamma1-beta), 4*gamma2 + 2)
100 msis = SIS.Parameters(
101     n=k*n,
102     q=q,
103     length_bound=zeta_,
104     m=2*k*n,
105     norm=+Infinity,
106     tag="SIS"
107 )
108
109 print('msis (strong unforgeability)')
110 print(msis)
111 SIS.estimate(msis)
112
113 # Compute (approximate) rejection probabilities and expected number
114   of tries
115 prob_z = ((2*(gamma1 - beta) - 1) / (2*gamma1)) ** (k*n)
116 prob_r0 = ((2*(gamma2 - beta) - 1) / (2*gamma2)) ** (k*n)
117 prob_acc = prob_z * prob_r0
118 E_tries = float(1/prob_acc)
119
120 print("E_tries = " + str(E_tries))
121
122 # Only second-preimage resistance is required
123 challenge_entropy = log(binomial(n, tau) * 2**tau, 2)
124 print("Challenge entropy = " + str(int(challenge_entropy)))
125
126 # Figure out the size of the prime for NTT operations
127 coefficient_size = 1
128 coefficient_size *= 2 # taking into account sign for y and z
129 if k > 1:
130     coefficient_size *= 3 # for contribution of (y+2)
131     coefficient_size *= q-1 # coefficient size of A
132     coefficient_size *= gamma1-1 # coefficient size of y and z
133     coefficient_size *= n # summing across polynomial
134     coefficient_size *= k # summing across matrix
135 print("max coefficient bit length = " + str(float(log(
136     coefficient_size, 2))))
137
138 # Compare the conservative bound for q/p*c*b0 against gamma2.
139 print("q/p*cb0: " + str(q/p*tau*2**((d-1))))
140 print("gamma2: " + str(gamma2))
141 print("conservative ||q/p*cb0||_inf bound < gamma2: " + str(q/p*
142     tau*2**((d-1)) < gamma2))
143
144 # Sizes
145 print("sk (bytes): " + str(int(64 + 64 + 128 + 2*k*n*ceil(log(2*eta
146     , 2))/8 + k*n*d/8)))
147 print("pk (bytes): " + str(int(64 + k*n*(ceil(log(p, 2)) - d)/8)))
148 print("sig (bytes, without h): " + str(int(64 + k*n*log(2*gamma1, 2)
149     /8)))

```

```

145
146 evaluate_params(sig128)
147 evaluate_params(sig256)
148 evaluate_params(sig512)

```

B Checking List

The following is the checking list prepared according to the submission requirements document, to facilitate locating the relevant contents.

- **1. Algorithm Description**

- **Complete algorithm operation processes:** Section 4, especially Algorithm 3.
- **Digital signature algorithms KeyGen, Sign, and Verify:** Section 4, especially Algorithm 3.
- **High-level description of key generation, signing, and verification:** Section 2.1.
- **Mathematical notation and basic definitions:** Section 3.1.
- **LWE/LWR background and formulas:** Section 3.2.
- **Radical-ring structure and formulas:** Sections 3.3 and 3.4.
- **Structured multiplication over the radical ring:** Section 3.4, especially Equation (6).
- **Algorithm-instance parameters for all security-strength levels:** Section 4, Table 2.
- **Parameters for Octarine-128, Octarine-256, and Octarine-512:** Section 4, Table 2.
- **Private-key, public-key, and signature sizes:** Section 4, Table 2.
- **Repetition values and security estimates in the parameter table:** Section 4, Table 2.
- **Sampling, randomness, seeds, hash/XOF use, and input/output handling:** Sections 3.1 and 4.
- **Tables supporting the algorithm description:** Section 2.3, Table 1; Section 4, Table 2; and Section 7, Tables 3–7.
- **Algorithm specification:** Section 4, especially Algorithm 3.

- **2. Design Rationale**

- **Main design ideas and rationale:** Sections 1, 2.1, and 5.
- **Mathematical hard problem:** Sections 2.3, 3.4, and 6.1.1.
- **RR-LWR as the underlying assumption:** Sections 1, 2.1, 3.4, and 6.1.1.
- **SIS-type assumption for unforgeability:** Sections 2.3 and 6.1.2.
- **Criterion for parameter selection:** Section 5.1 and Table 2.
- **Choice of radical-ring structure:** Sections 1, 2.1, 3.4, and 5.1.
- **Choice of n , k , p , q , η , d , τ , λ , γ_1 , γ_2 , β , and ω :** Table 2 and Section 5.1.
- **Use of power-of-two moduli and uniform sampling:** Sections 1, 2.1, 5.1, and 5.3.

- **Correctness and rejection-probability analysis:** Section 5.2.
- **Other design considerations:** Section 5.3.
- **Relation to existing work and comparative rationale:** Section 2.2.
- **3. Security Statements and Analyses**
 - **Security statements of the algorithm:** Sections 2.3 and 6.
 - **Theoretical security:** Section 6.1.
 - **RR-LWR hard problem:** Section 6.1.1.
 - **Unforgeability of Octarine:** Section 6.1.2.
 - **Security model and security proof:** Section 6.1.2, Theorems 4 and 5.
 - **Random-oracle-model security analysis:** Sections 2.3 and 6.1.2.
 - **Practical security:** Section 6.
 - **Complexity of known classical and quantum attacks for each instance:** Section 2.3, Table 1; Section 4, Table 2; and Appendix A.1.
 - **Security estimates for key recovery and unforgeability:** Section 2.3, Table 1; Section 4, Table 2; and Appendix A.1.
 - **BUFF properties:** Section 6.1.3.
 - **Reproducible security-estimation script:** Appendix A.1.
 - **Physical security:** Section 6.
- **4. Performance Evaluation**
 - **Performance analyses:** Sections 7.1 and 7.2.
 - **Arithmetic implementation analysis:** Section 7.1.
 - **Reference implementation arithmetic:** Section 7.1.
 - **AVX2 implementation arithmetic:** Section 7.1.
 - **Sampling and expansion cost analysis:** Section 7.2.
 - **Radical-ring product cost analysis:** Section 7.2.
 - **Operation-level performance analysis:** Section 7.2.
 - **Communication and storage cost analysis:** Table 2 and Section 7.2.
 - **Performance test results for all algorithm instances on popular 64-bit PC processors:** Section 7.3.
 - **Detailed configuration of implementation platforms:** Section 7.3, Experimental setup. The document specifies the CPU, clock rate, instruction sets, operating system, compiler, implementation language, and benchmark method.
 - **Whole-function performance results for KeyGen, Sign, and Verify:** Section 7.3, Tables 3 and 7.
 - **Reference and AVX2 performance results:** Section 7.3, Table 3.
 - **Arithmetic breakdown results:** Section 7.3, Tables 4–6.
 - **SHA3/SHAKE AVX2 whole-function performance results:** Section 7.3, Table 7.
 - **Comparison with Dilithium AVX2 implementation:** Section 7.3, Table 7.
- **5. Feature Statements**
 - **Feature statements, including innovativeness, simplicity, flexibility, compatibility, extensibility, and performance advantages on variant platforms:** Section 1.

References

- [ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 83–107. Springer, Berlin, Heidelberg, April / May 2002. [doi:10.1007/3-540-46035-7_6](https://doi.org/10.1007/3-540-46035-7_6).
- [AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 57–74. Springer, Berlin, Heidelberg, August 2013. [doi:10.1007/978-3-642-40041-4_4](https://doi.org/10.1007/978-3-642-40041-4_4).
- [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. [doi:10.1515/jmc-2015-0016](https://doi.org/10.1515/jmc-2015-0016).
- [APUPG17] Troncoso-Pastoriza Alberto Pedrouzo-Ulloa, Juan Ramón and Fernando Pérez-González. Multivariate cryptosystems for secure processing of multidimensional signals. *ArXiv*, abs/1712.00848, 2017.
- [BBE⁺18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Cham, April / May 2018. [doi:10.1007/978-3-319-78375-8_12](https://doi.org/10.1007/978-3-319-78375-8_12).
- [BCV20] Carl Bootland, Wouter Castryck, and Frederik Vercauteren. On the security of the multivariate ring learning with errors problem. In *ANTS-XIV*, volume 4, pages 57–71, 2020. [doi:10.2140/obs.2020.4.57](https://doi.org/10.2140/obs.2020.4.57).
- [BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO’98*, volume 1462 of *LNCS*, pages 26–45. Springer, Berlin, Heidelberg, August 1998. [doi:10.1007/BFb0055718](https://doi.org/10.1007/BFb0055718).
- [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, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 209–224. Springer, Berlin, Heidelberg, January 2016. [doi:10.1007/978-3-662-49096-9_9](https://doi.org/10.1007/978-3-662-49096-9_9).
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. [doi:10.1145/2090236.2090262](https://doi.org/10.1145/2090236.2090262).
- [BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Berlin, Heidelberg, April 2012. [doi:10.1007/978-3-642-29011-4_42](https://doi.org/10.1007/978-3-642-29011-4_42).
- [BRvV22] Joppe W. Bos, Joost Renes, and Christine van Vredendaal. Post-quantum cryptography with contemporary co-processors: Beyond Kronecker, Schönhage-Strassen & Nussbaumer. In *31st USENIX Security Symposium*

- (*USENIX Security 22*), pages 3683–3697, Boston, MA, 2022. USENIX Association. URL: <https://www.usenix.org/conference/usenixsecurity22/presentation/bos>.
- [BRVW26] Joppe W. Bos, Joost Renes, Frederik Vercauteren, and Peng Wang. Structured module lattice-based cryptography: Fine-grained parameter selection without compromising performance. *IACR Transactions on Cryptographic Hardware and Embedded Systems (to appear)*, 2026(3), 2026. URL: <https://eprint.iacr.org/2026/098>.
- [CCD⁺24] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Marc Möller, Damien Stehlé, and MinJune Yi. HAETAE: Shorter lattice-based Fiat-Shamir signatures. *IACR TCHES*, 2024(3):25–75, 2024. doi:10.46586/tches.v2024.i3.25-75.
- [CDF⁺20] Cas Cremers, Samed Düzlülü, Rune Fiedler, Marc Fischlin, and Christian Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. Cryptology ePrint Archive, Report 2020/1525, 2020. URL: <https://eprint.iacr.org/2020/1525>.
- [CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vадnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Berlin, Heidelberg, September 2014. doi:10.1007/978-3-662-44709-3_11.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_26.
- [Coo66] S. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966.
- [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.
- [DFH⁺24] Jelle Don, Serge Fehr, Yu-Hsuan Huang, Jyun-Jie Liao, and Patrick Struck. Hide-and-seek and the non-resignability of the BUFF transform. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 347–370. Springer, Cham, December 2024. doi:10.1007/978-3-031-78020-2_12.
- [DFHS24] Jelle Don, Serge Fehr, Yu-Hsuan Huang, and Patrick Struck. On the (in)security of the BUFF transform. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 246–275. Springer, Cham, August 2024. doi:10.1007/978-3-031-68376-3_8.
- [DKL⁺18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. URL: <https://tches.iacr.org/index.php/TCHES/article/view/839>, doi:10.13154/tches.v2018.i1.238-268.

- [DKR⁺20] Jan-Pieter D’Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [DS24] Samed Düzl  and Patrick Struck. The role of message-bound signatures for the beyond UnForgeability features and weak keys. Cryptology ePrint Archive, Report 2024/1669, 2024. URL: <https://eprint.iacr.org/2024/1669>.
- [KO62] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962. URL: <https://www.mathnet.ru/eng/dan26729>.
- [Kro82] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *Journal für die reine und angewandte Mathematik*, 92:1–122, 1882.
- [LDK⁺22] Vadim Lyubashevsky, L o Lucas, Eike Kiltz, Tancre de Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehl , and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_1.
- [LS15] Adeline Langlois and Damien Stehl . Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. doi:10.1007/s10623-014-9938-4.
- [Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_35.
- [Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Berlin, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_43.
- [MGTF19] Vincent Migliore, Beno t G rard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Val rie Gauthier-Uma a, Mart n Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, Lecture Notes in Computer Science, pages 344–362. Springer, 2019. doi:10.1007/978-3-030-21568-2_17.
- [NR23] Parker Newton and Silas Richelson. A lower bound for proving hardness of learning with rounding with polynomial modulus. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 805–835. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_26.

- [OTF⁺20] Hiroki Okada, Atsushi Takayasu, Kazuhide Fukushima, Shinsaku Kiyomoto, and Tsuyoshi Takagi. A compact digital signature scheme based on the module-LWR problem. In Weizhi Meng, Dieter Gollmann, Christian Damsgaard Jensen, and Jianying Zhou, editors, *ICICS 20*, volume 11999 of *LNCS*, pages 73–90. Springer, Cham, August 2020. doi:[10.1007/978-3-030-61078-4_5](https://doi.org/10.1007/978-3-030-61078-4_5).
- [Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016. doi:[10.1561/04000000074](https://doi.org/10.1561/04000000074).
- [PFH⁺22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [PN25] Rafaël Del Pino and Guilhem Niot. Finally! A compact lattice-based threshold signature. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part III*, volume 15676 of *LNCS*, pages 169–199. Springer, Cham, May 2025. doi:[10.1007/978-3-031-91826-1_6](https://doi.org/10.1007/978-3-031-91826-1_6).
- [Pol71] John M. Pollard. The fast Fourier transform in a finite field. *Mathematics of Computation*, 25(114):365–374, 1971.
- [PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Berlin, Heidelberg, May 2013. doi:[10.1007/978-3-642-38348-9_9](https://doi.org/10.1007/978-3-642-38348-9_9).
- [PUTPG⁺21] Alberto Pedrouzo-Ulloa, Juan Ramón Troncoso-Pastoriza, Nicolas Gama, Mariya Georgieva, and Fernando Pérez-González. Revisiting multivariate ring learning with errors and its applications on lattice-based cryptography. *Mathematics*, 9(8), 2021. URL: <https://www.mdpi.com/2227-7390/9/8/858>, doi:[10.3390/math9080858](https://doi.org/10.3390/math9080858).
- [PUTPPG15] Alberto Pedrouzo-Ulloa, Juan Ramón Troncoso-Pastoriza, and Fernando Pérez-González. Multivariate lattices for encrypted image processing. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1707–1711, 2015. doi:[10.1109/ICASSP.2015.7178262](https://doi.org/10.1109/ICASSP.2015.7178262).
- [PUTPPG16] Alberto Pedrouzo-Ulloa, Troncoso-Pastoriza, and Fernando Pérez-González. On ring learning with errors over the tensor product of number fields, 2016. URL: <https://arxiv.org/abs/1607.05244>.
- [RCDB24] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D’Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Trans. Embed. Comput. Syst.*, 23(2), March 2024. doi:[10.1145/3603170](https://doi.org/10.1145/3603170).
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. doi:[10.1145/1060590.1060603](https://doi.org/10.1145/1060590.1060603).
- [Reg10] Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204, 2010. doi:[10.1109/CCC.2010.26](https://doi.org/10.1109/CCC.2010.26).

- [Too63] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers, published in soviet math (translations of dokl. adad. nauk. sssr), 4, 1963.