

ARCANE-Mithril

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	2
2	Introduction	5
2.1	Overview of the Basic Approach	5
2.2	Relation to Existing Work	7
3	Preliminaries	11
3.1	Notation	11
3.2	Lattice-based Cryptography	12
3.2.1	Learning with Errors	12
3.2.2	Learning with Rounding	13
3.2.3	Security of Learning with Rounding	14
3.3	Rings as Structured Modules	15
3.4	Radical Ring-LWR	16
4	Algorithm Description	17
5	Security Statements and Analyses	24
5.1	Theoretical Security	24
5.1.1	IND-CPA security of Mithril.PKE	24
5.1.2	IND-CCA security of Mithril.KEM	25
5.1.3	Security of the $\mathbf{RR-LWR}$ Instance	25
5.1.4	Concrete Attack Estimate Methodology	27
5.2	Practical Security	29
5.2.1	Decryption failure attacks	29
5.2.2	Side-channel attacks	30
6	Performance Evaluation	30
6.1	Arithmetic implementations	30
6.2	Performance Analyses	31
6.3	Performance Test	32
A	SageMath Script	35
A.1	Security estimates	35
A.2	Failure Probability	37
B	Checking List	41

Errata and Revisions

Version 1 (30 June 2026) Updated submission of the ARCANE-Mithril specification and supporting documentation.

Version 1.1 (11 August 2026)

- A typo in Algorithm 1 function `Mithril.PKE.Enc` Line 2 was fixed, where $(\frac{p}{2 \cdot t} - \frac{q}{2 \cdot p})$ should have been $(\frac{q}{2 \cdot p} - \frac{p}{2 \cdot t})$ according to the standard Ring-LWR and Module-LWR definitions. This affects the decryption failure rate. The implementations have been updated accordingly.
- In the ARCANE-Mithril specification v1.0 it was stated that “*As was done previously for Saber-style estimates, we heuristically model the failure events for the shared secret bits as independent; we do not claim a strict proof of all coefficient correlations.*” Additionally, it was stated that “*the radical ring structure includes $k - 1 - i$ multiplications by $y + 2$ for the inner product of row i of $\mathbf{b}$ and $\mathbf{s}'$, see Equation (6).* This increases the norm of its coefficients, for which we include a worst-case factor 3 in the calculation of the failure probability.” After further modeling it turns out that the contribution of $y + 2$ creates correlations in the shared secret bits that increase the variance beyond the already included factor 3. We have updated the failure probability modeling in Section 5.2.1 to calculate the full distribution of $(y + 2)\mathbf{es}'$, no longer relying on heuristic assumptions beyond standard Ring-LWR. Since this increases the failure probability for Mithril-256 and Mithril-512, we have updated the parameter sets in Table 2 and the implementations accordingly.

Version 1.2 (22 September 2026)

- Update performance tests and results for the new versions of Mithril-256 and Mithril-512.

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 4. 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 KEM: ARCANE-Mithril (we also use the shorter Mithril).

2.1 Overview of the Basic Approach

The security of Mithril is based on the hardness of the Radical Ring Learning With Rounding (RR-LWR) problem. Informally, RR-LWR is the Learning With Rounding problem instantiated over the radical ring $\mathbf{S}_{q,n,k}$, where arithmetic is performed in a structured ring that extends a power-of-two cyclotomic ring. Given uniformly random public data derived from $\mathbf{S}_{q,n,k}$, the RR-LWR problem asks to distinguish rounded linear products involving a secret element from uniformly random values, or to recover the secret itself. The rounding is applied coefficient-wise in the transported monomial basis, meaning the image of the radical-ring monomial basis under the number-field isomorphism of Theorem 4. Thus the assumption is a Ring-LWR variant over the radical number field rather than the canonical-embedding form of Ring-LWR. The ring $\mathbf{S}_{q,n,k}$

contains the cyclotomic sub-ring $\mathbf{R}_{q,n}$, which enables efficient arithmetic while preserving the algebraic properties required for LWR-based constructions. From a lattice perspective, RR-LWR can be compared with an LWR instance over a lattice of dimension proportional to nk , where the parameter k provides an additional degree of freedom for adjusting security levels and the concrete estimates account for the resulting basis distortion.

No new cryptographic assumptions beyond RR-LWR are introduced in the construction of Mithril. The key encapsulation mechanism is obtained from the public-key encryption scheme via a Fujisaki–Okamoto style transform, and its security relies on the assumed hardness of RR-LWR together with the standard properties of the employed hash functions.

Choice of ring structure and parameter selection criteria. The design of Mithril is guided by the goal of enabling flexible parameter selection while retaining the simplicity and efficiency of LWR-based constructions. To this end, the scheme is instantiated over the radical ring $\mathbf{S}_{q,n,k}$, where n is the base-ring degree and k is the radical extension degree.

Varying k while keeping the power-of-two base-ring degree n fixed changes the total algebraic degree nk , which is the effective lattice dimension used in the generic attack estimates. This provides fine-grained security scaling without changing the underlying cyclotomic degree. The subring $\mathbf{R}_{q,n}$ is a power-of-two cyclotomic ring, supporting efficient polynomial arithmetic and compatibility with established implementation techniques for LWR-based schemes.

The moduli p , q , and t are chosen as powers of two, and their bit-lengths are denoted by ϵ_p , ϵ_q , and ϵ_t , respectively. This choice simplifies rounding and scaling operations in the LWR construction, as all such operations can be implemented as coefficient-wise shifts and truncations. The relative sizes of these moduli are selected to ensure correct decryption with high probability while providing sufficient separation between noise and message components.

Finally, the parameter ℓ determines the number of coefficients extracted from a ring element during encryption and decryption. By operating on only the ℓ highest-degree coefficients in x , the scheme avoids unnecessary computation while still providing a message space of the desired size. The selection of ℓ is coordinated with the choices of n , k , and the rounding parameters to ensure both correctness and security.

We refer to Table 2 for a full overview of the parameter sets.

Failure probability. The correctness of the public-key encryption scheme underlying Mithril relies on the correctness of the underlying LWR construction. Due to the use of rounding and coefficient truncation, decryption is not perfectly correct and may fail with a small probability.

During encryption, the message is embedded into the ciphertext by adding a scaled message component to a rounded linear product. Decryption recovers the message by reversing the scaling and rounding operations. A decryption failure occurs when the accumulated rounding error causes a coefficient to cross a decision boundary, leading to an incorrect decoded bit.

The sources of decryption error are twofold. First, the LWR rounding operation introduces an approximation error whose magnitude depends on the ratio between the moduli p and q . Second, only the ℓ highest-degree coefficients in x are used for message encoding and decoding, which restricts the available error margin. These effects are controlled by the choice of the parameters ϵ_p , ϵ_q , ϵ_t , and ℓ .

For the parameter sets specified in this document, the decryption failure probability is estimated to be negligible, typically of the form $2^{-\lambda}$ for a security parameter λ . This probability is accounted for in the design of the key encapsulation mechanism: in the Fujisaki–Okamoto transform [FO99], decryption failures enter the security bound as an additive term.

As a result, while the encryption scheme is not perfectly correct, the estimated probability of failure is sufficiently small for the intended security levels and is included in the overall security accounting of Mithril when used within the specified KEM construction.

We refer to Section 5.2.1 for a more detailed analysis of the failure probability.

Sampling. The scheme makes extensive use of deterministic sampling from binary seeds for generating public matrices and secret values. This design simplifies key storage and transmission, enables compact representations of public parameters, and supports reproducible behavior across implementations. All randomness used in the scheme is explicitly derived from seeds using well-defined expansion functions.

Partial ring elements. The use of partial ring elements through the parameter ℓ allows the scheme to avoid unnecessary computation. By restricting message encoding and decoding to the ℓ highest-degree coefficients in x , the scheme reduces computational overhead while preserving the required message space and correctness guarantees. This design choice also facilitates efficient implementations without altering the underlying security assumptions, and reduces the failure probability compared to a full ring element.

Fujisaki-Okamoto transformations. The structure of the public-key encryption scheme is compatible with standard transformations from IND-CPA secure encryption to IND-CCA secure key encapsulation mechanisms. In particular, the encryption algorithm admits a formulation with explicit randomness, which enables the application of a Fujisaki-Okamoto style transform [FO99] without modifying the core arithmetic of the scheme.

High-assurance implementation. The algorithm specification is designed to support constant-time implementations and to simplify secure implementation with respect to side-channel and fault attacks. All arithmetic operations are defined in a coefficient-wise manner, and conditional behavior in the key encapsulation mechanism can be implemented using constant-time selection primitives. These considerations help reduce the risk of side-channel leakage in practical deployments.

2.2 Relation to Existing Work

We compare Mithril to the most relevant existing lattice-based KEMs. Our 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. For a detailed comparison to LWR-based proposals we refer to Table 1.

CRYSTALS-Kyber / ML-KEM. The most significant difference between Mithril and ML-KEM (formerly known as CRYSTALS-Kyber) is that Mithril is based on Ring-LWR, whereas ML-KEM is based on Module-LWE. We refer to Section 3 for a detailed recollection of the relation between the underlying hard problems.

A notable difference compared to many LWE-based designs, where q can be selected to be a prime, is that the Number-Theoretic Transform (NTT) does not apply natively due to the lack of appropriate roots of unity in $\mathbf{Z}_q$. This affects the efficiency of polynomial arithmetic in $\mathbf{R}_q$, leading to generally slower implementations of LWR than LWE. However, it was first shown in [CHK⁺21] that the asymptotically faster NTTs can still be used by embedding $\mathbf{R}_q$ into a larger NTT-friendly ring with prime modulus q' . While this would increase the bit length of the modulus by at least a factor of 2, it can still lead to improved arithmetic compared to Toom-Cook or Karatsuba on certain platforms. For example, [CHK⁺21] uses 6 layers of size-256 negacyclic NTTs followed by 4×4 schoolbook multiplications using

$$q' = 25166081 = 196610 \cdot 128 + 1$$

Scheme	$f(x)$	k	ℓ	ϵ_q	ϵ_p	ϵ_t	$D_{\mathbf{A}}$	$D_{\mathbf{s}}$	B
Mithril-128	$(x^k - 2)^{128} + 1$	5	1	13	11	3	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
Mithril-256	$(x^k - 2)^{128} + 1$	9	2	13	11	3	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
Mithril-512	$(x^k - 2)^{128} + 1$	17	4	13	11	8	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
LightSaber [DKRV18]	$x^{256} + 1$	2	1	13	10	3	$\mathcal{U}_{\epsilon_q}$	β_5	1
Saber [DKRV18]	$x^{256} + 1$	3	1	13	10	4	$\mathcal{U}_{\epsilon_q}$	β_4	1
FireSaber [DKRV18]	$x^{256} + 1$	4	1	13	10	6	$\mathcal{U}_{\epsilon_q}$	β_3	1
uLightSaber [DKR ⁺ 20, Sec. A.2]	$x^{256} + 1$	2	1	12	10	3	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
uSaber [DKR ⁺ 20, Sec. A.2]	$x^{256} + 1$	3	1	12	10	4	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
uFireSaber [DKR ⁺ 20, Sec. A.2]	$x^{256} + 1$	4	1	12	10	6	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_2$	1
Florete [MKKV21, KNK ⁺ 25]	$x^{768} - x^{384} + 1$	1	1	10	9	3	$\mathcal{U}_{\epsilon_q}$	β_4	1
Espada [MKKV21, KNK ⁺ 25]	$x^{64} + 1$	12	1	15	13	3	$\mathcal{U}_{\epsilon_q}$	β_4	4
LightSable [MKKV21, KNK ⁺ 25]	$x^{256} + 1$	2	1	11	9	2	$\mathcal{U}_{\epsilon_q}$	β_1	1
Sable [MKKV21, KNK ⁺ 25]	$x^{256} + 1$	3	1	11	9	4	$\mathcal{U}_{\epsilon_q}$	β_1	1
FireSable [MKKV21, KNK ⁺ 25]	$x^{256} + 1$	4	1	11	9	2	$\mathcal{U}_{\epsilon_q}$	β_1	1
TiMER* [HCCSS24]	$x^{256} + 1$	2	1	10	8	3	$\mathcal{U}_{\epsilon_q}$	spCBD	2
SMAUG-T128* [HCCSS24]	$x^{256} + 1$	2	1	10	8	5	$\mathcal{U}_{\epsilon_q}$	spCBD	1
SMAUG-T192* [HCCSS24]	$x^{256} + 1$	3	1	10	8	8	$\mathcal{U}_{\epsilon_q}$	spCBD	1
SMAUG-T256* [HCCSS24]	$x^{256} + 1$	5	1	10	8	6	$\mathcal{U}_{\epsilon_q}$	spCBD	1

Table 1: Comparison of parameters for Ring / Module LWR-based post-quantum cryptographic schemes. All schemes except **Mithril** use Module-LWR parameters with ring $\mathbf{R} = \mathbf{Z}[x]/(f(x))$, matrix distribution $D_{\mathbf{A}}$, and (ephemeral) secret-key distribution $D_{\mathbf{s}}$ for $\mathbf{s}$ and/or $\mathbf{s}'$. The column B gives the number of bits per input-message coefficient. Here, ϵ_q , ϵ_p , and ϵ_t denote $\log_2 q$, $\log_2 p$, and $\log_2 t$, respectively. For the Module-LWR schemes, k denotes the module rank; for **Mithril**, the same table column reports the radical extension degree.

* Uses *LWE* for key generation and *LWR* for *Encaps/Decaps*.

for the Saber and FireSaber parameter sets resulting in approximately 60 percent faster multiplication than in $\mathbf{R}_{8192}$. Ultimately, while there is certainly a measurable difference in the efficiency of LWR and LWE, they are close enough in performance that it is not significant for many practical applications.

Additionally, while **Mithril** is based on Ring-LWR as opposed to a *module* problem, the ring $\mathbf{S}_{q,n,k}$ inherits the benefits from Module-LWR (flexible parameter selection) while retaining most of the performance advantages from Ring-LWR, by tuning the parameter k . By using a radical ring we reduce the size of the public matrix, which directly reduces the cost of sampling as well as the memory requirements. 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 and no (pseudo-)randomly generated bits need to be discarded.

For high-assurance implementations, LWR-based schemes, like **Mithril**, benefit from two key advantages:

- Power-of-two moduli simplify mask conversions.

- Rounding is easier to mask than noise sampling.

These properties motivated the first-order masked Saber framework, achieving a $2.5\times$ overhead on ARM Cortex-M4, compared to $5\times$ for LWE-based schemes. Higher-order masked implementations of Saber and uSaber [KDvB⁺22] show that LWR schemes scale better than LWE counterparts with increasing t . Using uniform secret distributions is the most efficient approach since it simplifies sampling. The masked implementations of LWR-based KEMs Florete, Espada, and Sable from the suite, Scabbard, also perform better than ML-KEM due to the power-of-two moduli, and these schemes are portrayed as masking-friendly schemes in [KKV23]. Masked Florete and masked Sable are even faster than masked Saber for any order masking due to their efficient parameter choices. Although uSaber is shown as a mask-friendly version of Saber in [KDvB⁺22], Sable performs better than uSaber, and this advantage remains after masking, as masked Sable shows slight improvement over uSaber in performance for any order masking. The Mithril design benefits from these advantages in the same way.

In hardware, [FVR⁺22] proposed RISC-V instruction set extensions for masking Kyber (ML-KEM) and Saber. Their results confirm LWR masking efficiency over LWE schemes. [ABC⁺23] further optimized masked polynomial multiplication for power-of-two moduli. Their Saber hardware achieves first-order resistance with only $1.7\times$ area overhead and completes multiplication in just 340 cycles: significantly outperforming generic masking approaches.

Finally, we take a more conservative design approach for the 128-bit parameter set and select $n = 128$ with $k = 5$ to work with a lattice of dimension 640. Concretely, this increases the bit security from 140 for ML-KEM 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 M-LWE/LWR, so we mitigate some of the impact on the cost of matrix generation. While we still increase the key and ciphertext sizes and impact overall performance compared to a smaller lattice, we believe this to be acceptable to ensure one has more confidence in the achieved 128-bit security level.

(u)Saber. The Saber scheme [DKRV18] is the first KEM that is fully based on (module) LWR. It was submitted to the NIST PQC competition in 2016 and made it to the third round [DKR⁺20]. By fully relying on LWR no sampling from an error distribution is needed. The Saber design aims for simplicity and efficiency. The choice of ring $\mathbf{Z}_q[x]/(x^{256} + 1)$ with $q = 2^{13}$ is fixed across all parameter sets, varying instead the module dimension, ciphertext modulus and secret key distribution, and the arithmetic modulo a power of two q allows for straightforward implementation without modular reduction or rejection sampling. The NIST proposal defines three parameter sets LightSaber, Saber and FireSaber corresponding to NIST security level 1, 3 and 5 respectively. Secret keys are selected from a binomial distribution β_η with $\eta = 3, 4, 5$. A drawback of the choice of q is that NTTs are not natively supported in $\mathbf{Z}_q$ and the implementations therefore initially relied on Toom-Cook. However, by embedding into a larger ring with appropriately selected modulus (about twice the bit length of q) the polynomial multiplication can still be implemented with NTTs, albeit not quite as efficient as for example in ML-KEM.

Uniform Saber (uSaber) [DKR⁺20, Section A.2] is a variant of Saber proposed in the Round 3 submission of Saber that is not part of the main submission. The main difference is that the secret vectors $\mathbf{s}$ and $\mathbf{s}'$ are sampled from the uniform distribution $\mathcal{U}_2$ instead of the binomial distribution β_η . This means that the secret coefficients are in the interval $[-2, 1]$ instead of the interval $[-\eta, \eta]$ in Saber and can be represented with exactly 2 bits each.

The main advantage of uSaber is its simplified secret generation process, which is used during key generation, encryption, and consequently in encapsulation and decapsulation for a KEM. A key benefit is the significant reduction in the amount of (pseudo-)random bits needed to achieve a comparable secret variance, and thus a similar security level. For instance, uSaber parameter sets require only two random bits per secret coefficient, compared to the six to ten

bits required by standard Saber when using binomial sampling. This leads to faster hashing, which in turn reduces both computational overhead and the attack surface. Since hashing is a highly non-linear process, this reduction is especially valuable for masked implementations.

Furthermore, uSaber eliminates the need to compute a binomial distribution. This simplification makes implementations more straightforward, particularly in masked settings where binomial sampling, being non-linear, adds considerable complexity. By removing it, uSaber not only enhances efficiency but also lowers the attack surface, resulting in a more secure and practical design.

The Mithril design follows the design principles of uSaber and extends them to the RR-LWR setting. We acknowledge the benefits of (small) uniform distributions for simple and secure implementation, including for protection against side-channel attacks. The use of RR-LWR allows us to improve efficiency compared to uSaber by reducing the size of the matrix (more precisely, by replacing the matrix with a ring element) and to increase the security level more granularly. By using $n = 128$ as opposed to $n = 256$ combined with $k = 5$, we achieve a 128-bit security level that is strictly higher than uSaber.

Scabbard Suite: Florete, Espada and Sable. Building upon the improvements of LWR-based schemes in many aspects such as multiplication routines, cryptanalysis, parameter selection, etc., Scabbard [MKKV21, KNK⁺25] was proposed during the course of the NIST standardization effort as a suite of three KEMs. Each of the three KEMs is tailored to improve the state-of-the-art in different aspects such as speed, area requirements, memory footprint, and parallel implementation.

The first scheme “Florete” is a Ring-LWR scheme. As opposed to Saber, which uses module rank $k = 3$ and polynomial degree $n = 256$ over $\mathbf{Z}_q$ for $q = 2^{13}$, Florete relies on the ideal lattice defined by the ring $\mathbf{Z}_q[x]/(x^{768} - x^{384} + 1)$ for $q = 2^{10}$ and fixes the module rank to be $k = 1$. Aiming for the 192-bit security level of Saber, this reduces the matrix dimension of $\mathbf{A}$ from 3×3 to a single ring element which removes the need for matrix-vector products. An advantage of this approach is that fewer pseudo-random bytes are required for the generation of $\mathbf{A}$, which can be a performance bottleneck (particularly in software). The design is done in such a way that the multiplication routines from Saber can be fully re-used. A downside of this design is that the security compared to Saber is reduced by approximately 15 bits [MKKV21, Table 2]. A similar approach is followed to match the security levels of LightSaber and FireSaber using rings with polynomial degree 512 and 1024 respectively, as proposed in a later work [KNK⁺25]. While Florete, like Mithril, also uses a ring, its ring is not scalable to different security levels and therefore aims only to achieve a 192-bit security level.

The second scheme is called “Espada” and aims to improve efficiency by allowing for more parallelization in the algorithm. The main idea is to reduce the larger polynomial multiplication of 256 coefficients into multiple smaller multiplications consisting of 64 coefficients each. Although a 256×256 polynomial multiplication can be reduced to 64×64 polynomial multiplications through Toom-Cook or Karatsuba, the use of such algorithms does not enable the same level of parallelization and also increases the size of the multiplier due to the evaluation and interpolation stages of Toom-Cook and Karatsuba. By making $n = 64$ the native polynomial degree of the underlying lattice, the implementation cost of the base multiplication routine is reduced, particularly in hardware [MTK⁺20] and microcontrollers. It also improves the granularity of the parameter selection. As a result, the module rank k is increased from 3 to 12 to achieve a 128-bit security level (and similarly for other security levels). While this increases the total number of 64×64 polynomial multiplications that are required compared to Saber, their independence makes parallel implementation more straightforward. The reduction of n from 256 to 64 requires the selection of $B = 4$ bits per coefficient, affecting the failure probability and ultimately requiring an increase in the size of q to 15 bits. Mithril differs from Espada by being a ring-based KEM. While aiming to achieve similar advantages as Espada by reducing n

from 256 to 128, Mithril does not need to select multiple bits per coefficient. Selecting $B = 1$ for Mithril avoids any impact on failure probability.

The final scheme “Sable” follows the design of Saber most closely, but changes the secret key distribution. Whereas LightSaber, Saber and FireSaber change their choice of η to select secret key coefficients in the range from the centered binomial distribution β_η in $[-\eta, \eta]$ for $\eta = 5, 4, 3$ respectively, Sable fixes $\eta = 1$ across all parameter sets, which makes switching between different schemes more seamless. In this case, only 2 bits are required to store a secret key coefficient, making the storage requirement equal to uSaber and helping with masked implementations. Overall, this change reduces the complexity of the binomial multiplier and makes the multiplier design uniform across parameter sets if it is implemented using the maximum values of p and q . To make up for the loss in security the values of p and q are decreased accordingly. The paper remarks that the selection of p and q for Saber used erroneous estimations as it was considering a *continuous* rather than a *discrete* uniform distribution in the rounding error. The LightSable, Sable and FireSable parameter sets all assume the discrete uniform distribution which enables more accurate parameter selection and security estimations. Mithril also aims for small secret key coefficients, but allows a larger interval for the secret key than Sable, and restricts itself to uniform distributions. This simplifies the sampling procedures and makes more efficient use of each sampled bit.

SMAUG and Friends. SMAUG-T [HCCSS24], a merger of SMAUG [CCHY24] and TiGER [PJP⁺22], is the most recent LWR-based proposal and has been selected as a winner in the Korean PQC standardization effort. Its key generation routine reduces to LWE while the Encaps/Decaps rely on LWR. For the generation of the ephemeral encapsulation secret they use a sparse centered binomial distribution, more precisely providing a ternary value which is zero with probability $1 - 2r$, -1 with probability r and 1 with probability r for $0 \leq r < 1/2$. The value r is selected as $1/8$ or $3/16$, where the power of two in the denominator is specifically chosen to ease implementation. It should be noted that the design has been updated in between v2.0 and v4.0 as a result of non-constant-time implementation of some of its samplers, so careful implementation is required. The advantage of the sparse secrets is that they reduce decryption failures and improve the efficiency of the KEM. In v3.0 the new “TiMER” parameter set was added to the specification, aiming to further lower the decryption failure probability due to the D2 error reconciliation mechanism. By encoding each message bit into two polynomial coefficients the probability of failure reduces at the cost of additional encoding/decoding.

Mithril does not use an LWE assumption for key generation and does not use sparse distributions for private keys. We aim to reduce implementation complexity, in particular with respect to timing attacks and other side-channel attacks. While sparse secrets may offer advantages for performance, Mithril remains conservative and focuses on security first.

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

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

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 Mithril key encapsulation mechanism (KEM). The key encapsulation mechanism Mithril.KEM is derived from Mithril.PKE using an implicit-rejection Fujisaki–Okamoto transform. Mithril.PKE consists of the three algorithms KeyGen, Enc, and Dec, while Mithril.KEM consists of KeyGen, Encaps, and Decaps, as specified in Algorithm 1.

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. 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.

	Mithril-128	Mithril-256	Mithril-512
n	128	128	128
k	5	9	18
ℓ	1	2	4
ϵ_q	13	13	14
ϵ_p	11	11	12
ϵ_t	3	4	6
$D_{\mathbf{A}}$	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_{\epsilon_q}$	$\mathcal{U}_{\epsilon_q}$
$D_{\mathbf{s}}$	$\mathcal{U}_2$	$\mathcal{U}_2$	$\mathcal{U}_2$
KEM secret key (bytes)	1 136	2 000	4 224
KEM public key (bytes)	944	1 648	3 520
KEM ciphertext (bytes)	928	1 712	3 840
Shared secret (bytes)	16	32	64
Failure probability ($\log_2$)	-603	-278	-569
Core-SVP (Classical) (bits)	129	257	524
Core-SVP (Quantum) (bits)	117	233	476

Table 2: An overview of the three parameter sets of Mithril.

Algorithm 1 Mithril Public-Key Encryption (PKE) and Key Encapsulation Mechanism (KEM) algorithms.

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

- 1: $\text{seed}_A, \text{seed}_s \leftarrow \{0, 1\}^{512}$
- 2: $A \leftarrow D_A(S_{q,n,k}; \text{seed}_A)$
- 3: $s \leftarrow D_s(S_{q,n,k}; \text{seed}_s)$
- 4: $b = \lfloor \frac{p}{q} As \rfloor \in S_{p,n,k}$
- 5: **return** $(pk = (\text{seed}_A, b), sk = s)$

$c \leftarrow \text{Mithril.PKE.Enc}(pk = (\text{seed}_A, b), m)$

- 1: $\text{seed}_{s'} \leftarrow \{0, 1\}^{512}$
- 2: $s' \leftarrow D_s(S_{q,n,k}; \text{seed}_{s'})$
- 3: $A \leftarrow D_A(S_{q,n,k}; \text{seed}_A)$
- 4: $b' = \lfloor \frac{p}{q} \cdot (As') \rfloor \in S_{p,n,k}$
- 5: $v' = (bs')^{[k-\ell:k-1]} + \frac{q}{2 \cdot p} \in \mathbf{R}_{p,n}^\ell$ ▷ Select ℓ top coefficients of ring element
- 6: $c_m = \lfloor \frac{t}{p} \cdot ((v' + \frac{p}{2} \cdot m) \bmod p) \rfloor \in \mathbf{R}_{t,n}^\ell$
- 7: **return** $c = (c_m, b')$

$m' \leftarrow \text{Mithril.PKE.Dec}(sk = s, c = (c_m, b'))$

- 1: $v = (b's)^{[k-\ell:k-1]} \in \mathbf{R}_{p,n}^\ell$ ▷ Select ℓ top coefficients of ring element
- 2: $m' = \lfloor \frac{2}{p} \cdot ((v - \frac{p}{t} \cdot c_m + (\frac{q}{2 \cdot p} - \frac{p}{2 \cdot t})) \bmod p) \rfloor \in \mathbf{R}_{2,n}^\ell$
- 3: **return** m'

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

- 1: $(pk_{\text{pke}}, sk_{\text{pke}}) \leftarrow \text{Mithril.PKE.KeyGen}$
- 2: $z \leftarrow \{0, 1\}^{\ell n}$ ▷ Secret fallback value
- 3: $hpk \leftarrow \mathcal{F}_{H-F}(pk_{\text{pke}}, \lambda/8)$ ▷ Hash of public key
- 4: **return** $(pk = pk_{\text{pke}}, sk = (sk_{\text{pke}}, pk_{\text{pke}}, hpk, z))$

$(c, ss) \leftarrow \text{Mithril.KEM.Encaps}(pk)$

- 1: $m \leftarrow \{0, 1\}^{\ell n}$ ▷ Encode/Map to a message in $\mathbf{R}_{2,n}^\ell$
- 2: $hpk \leftarrow \mathcal{F}_{H-F}(pk, \lambda/8)$
- 3: $(ss, \text{seed}_{s'}) \leftarrow \mathcal{G}_{H-G}(hpk, m, \lambda/8 + 64)$ ▷ Derive key material and encryption seed
- 4: $c \leftarrow \text{Mithril.PKE.Enc}(pk, m; \text{seed}_{s'})$ ▷ Encryption using explicit randomness $\text{seed}_{s'}$
- 5: **return** (c, ss)

$ss \leftarrow \text{Mithril.KEM.Decaps}(sk, c)$

- 1: $m' \leftarrow \text{Mithril.PKE.Dec}(sk_{\text{pke}}, c)$
 - 2: $(ss', \text{seed}_{s'}) \leftarrow \mathcal{G}_{H-G}(hpk, m', \lambda/8 + 64)$ ▷ Recompute key material and encryption seed
 - 3: $c' \leftarrow \text{Mithril.PKE.Enc}(pk, m'; \text{seed}_{s'})$ ▷ Re-encrypt deterministically
 - 4: $\bar{s}s \leftarrow \mathcal{H}_{H-H}(c, z, \lambda/8)$
 - 5: **if** $c = c'$ **then** ▷ Implement in constant time
 - 6: $ss \leftarrow ss'$
 - 7: **else**
 - 8: $ss \leftarrow \bar{s}s$
 - 9: **end if**
 - 10: **return** ss
-

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

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

where q and n are powers of two, and k is an arbitrary positive integer. This is the same ring that is used by **Octarine**. There exists a straightforward isomorphism $\mathbf{S}_{q,n,k} \cong \mathbb{Z}_q[x]/((x^k - 2)^n + 1)$ [BRVW26, Section 4]. The scheme uses RR-LWR with coefficient-wise rounding in the radical-ring monomial basis; under this isomorphism, this becomes a transported-basis Ring-LWR instance over the associated number field. The ring $\mathbf{S}_{q,n,k}$ is a radical extension of degree k over the $2n$ -th cyclotomic base ring. Standard implementation techniques, such as NTTs or other polynomial-multiplication routines, can therefore be used if desired. The compact representation by k base-ring components replaces the quadratic public-matrix representation of a rank- k module construction.

We fix the ring modulus $q = 2^{13}$ and the base-ring degree $n = 128$ across the parameter sets, and vary the radical extension degree k , and hence the total algebraic degree nk , to obtain the required security levels. Although the ring is compatible with NTT-style implementations, the use of an NTT is optional. The RR-LWR rounding step maps coefficients from modulus q to the rounding modulus p . The scale factor is q/p , and the resulting deterministic quantization error is bounded in magnitude by $q/(2p)$. We select p such that $\mathbf{s}$ and $\mathbf{s}'$ follow the same uniform distribution over $[-\eta, \eta - 1]$, where $\eta = q/(2p) = 2$. Each coefficient of $\mathbf{s}$ and $\mathbf{s}'$ can therefore be represented with exactly $\log_2(2\eta) = 2$ bits.

Message space. In the setting of Module-LWE or Module-LWR, the message is embedded as an element of $\mathbf{R}_{2,n}$. As this consists of n coefficients of a single bit each, it naturally encodes n -bit messages. For Ring-LWR the message is encoded as an element of $\mathbf{S}_{2,n,k}$ instead, allowing nk -bit messages to be encoded. Since we do not need the full kn -bit message space, we embed our messages in the subspace where ℓ of the coefficients are non-zero, and the other $k - \ell$ coefficients are 0. As $n = 128$, we choose $\ell = 1, 2, 4$ for the respective parameter sets to make sure that $n\ell = \lambda$.

To optimize the performance of the scheme, we select the ℓ *most* significant coefficients of the ring element. This minimizes the number of reductions modulo $x^k - (y + 2)$ and therefore saves some multiplications. In the implementation, all multiplications by zero coefficients can be omitted. More precisely, we use the notation $\mathbf{g}^{[k-\ell:k-1]}$ for the polynomial $\mathbf{g} = \sum_{i=0}^{k-1} \mathbf{g}_i(y)x^i \in \mathbf{S}_{2,n,k}$ where $\mathbf{g}_0(y), \dots, \mathbf{g}_{k-\ell-1}(y) = 0$, which we interpret as a vector in $\mathbf{v} \in \mathbf{R}_{2,n}^\ell$ such that $\mathbf{v}_i = \mathbf{g}_{k-1-i}$. Thus the packed message and ciphertext components are serialized in descending x -degree order: the first $\mathbf{R}_{2,n}$ component is $\mathbf{g}_{k-1}(y)$, followed by $\mathbf{g}_{k-2}(y), \dots, \mathbf{g}_{k-\ell}(y)$.

Hashing. Mithril requires hash functions, or more specifically eXtensible Output Functions (XOFs), for instantiating random oracles as well as generating pseudo-random bits for key material and ephemeral secrets.

First, we use the functions $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ to instantiate the random oracles. As usual, we include the output length in bytes in the call to the hash function. The implementation realizes these roles with an explicit domain encoding before the underlying XOF is called. The encoded input is the concatenation of the header fields below, followed by the $\mathbf{nf}$ variable-length fields $|X_1| \parallel X_1 \parallel \dots \parallel |X_{\mathbf{nf}}| \parallel X_{\mathbf{nf}}$, where each field length $|X_i|$ is encoded as an 8-byte little-endian integer. All header integers are encoded little-endian as well.

Field	Value	Bytes
Protocol prefix	ARCANE-Mithril-v1	17
Label length	label	1
Label	label	label
Security level	λ	2
Module rank	k	1
Output length	outlen	4
Field count	nf	1

Equivalently, the label length, module rank, and field count are encoded as one byte each; the security level is a 16-bit little-endian integer, the output length is a 32-bit little-endian integer, and each variable-field length is a 64-bit little-endian integer. Here **outlen** is the requested output length in bytes. The labels are H_F , H_G , and H_H for the three KEM random-oracle roles. Specifically,

- *Public key digest*: $hpk \leftarrow \mathcal{F}_{H_F}(pk_{\text{pke}}, \lambda/8)$ creates a digest of the public key that can be stored as part of the private key.
- *Deterministic encryption*: $(ss, \text{seed}_{s'}) \leftarrow \mathcal{G}_{H_G}(hpk, m, \lambda/8 + 64)$ creates the shared secret and deterministic coins for the public-key encryption based on the public key and message.
- *Shared secret generation*: $\bar{s}s \leftarrow \mathcal{H}_{H_H}(c, z, \lambda/8)$ generates the shared secret to be used if the ciphertext is implicitly rejected.

For both the reference implementation and the optimized implementation submitted with this specification, all hash functions are instantiated with **pseudoXOF**. To enable a fair performance comparison with state-of-the-art optimized implementations, we additionally provide a SHAKE-based benchmarking configuration, in which the relevant hash functions are instantiated with **SHAKE256**. Since **SHAKE256** provides at most 256 bits of classical security due to its 512-bit capacity, this SHAKE-based configuration is not used for the 512-bit parameter set.

Second, we also require XOFs for pseudo-random bit generation to expand private and public key material from seeds. This is used in the generation of $\mathbf{s}$, $\mathbf{A}$ and $\mathbf{s}'$. We use XOF_A for the XOF on public data $\mathbf{A}$, and XOF_S for the XOF used for secret data. These high-volume XOF inputs use a compact length-delimited encoding:

$$\begin{aligned} \text{AMX1} \parallel \text{id} \parallel \text{IntegerToBytes}(\lambda, 2) \parallel \text{IntegerToBytes}(k, 1) \parallel \text{IntegerToBytes}(\text{outlen}, 2) \\ \parallel \text{IntegerToBytes}(|s|, 1) \parallel s \parallel \text{IntegerToBytes}(i, 1) \parallel \text{IntegerToBytes}(\text{lane}, 1). \end{aligned}$$

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 lane is the lane index. For XOF_A , the fields are the seed, a one-byte polynomial index, and a one-byte lane index. For XOF_S , the fields are the seed, the index byte 0, and the lane byte 0; a single stream generates all k secret polynomials. The seed length is encoded in one byte and the output length as a 16-bit little-endian integer; these bounds cover every implemented parameter set. We instantiate both XOF_A and XOF_S with **pseudoXOF** in implementation. For the fair performance comparison optimized implementation with state-of-the-art optimized implementations, we use **SHAKE128** for XOF_A and **SHAKE256** for XOF_S .

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. For large coefficients, e.g., with $\log_2(q) = 13$ -bit coefficients, we generate each of the k polynomials with n coefficients each independently using domain separation. This allows parallel generation of

the ring element polynomials. For ring elements with smaller coefficients, such as for $\mathbf{s}$ with 2-bit coefficients, only $2 \cdot 128/8 = 32$ bytes per polynomial are required. Therefore we choose to generate the full ring element from a single byte stream in such cases.

- $D_{\mathbf{A}}(\mathbf{S}_{q,n,k}, \text{seed}_{\mathbf{A}})$: Deterministically expands the seed $\text{seed}_{\mathbf{A}}$ 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 $\epsilon_q = 13$ 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 $\text{seed}_{\mathbf{A}}$. For each polynomial we use the index i to domain separate $\text{seed}_{\mathbf{A}}$. That is, the byte stream for $\mathbf{A}_i(y)$ is generated from the seed $\text{seed}_{\mathbf{A}}$ as the $13 \cdot 128/8 = 208$ -byte stream $B_i = \text{XOF}_A(\text{seed}_{\mathbf{A}}, i, 0)$. Second, we set $\alpha_{i,j} = B_i[13j : 13(j+1)]$, where $B_i[a : b]$ denotes bit index a to b , including a and excluding b .
- $D_{\mathbf{s}}(\mathbf{S}_{q,n,k}, \text{seed}_{\mathbf{s}})$: 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 $\text{seed}_{\mathbf{s}}$ is used by the XOF function XOF_S to output the byte stream $B = \text{XOF}_S(\text{seed}_{\mathbf{s}}, 0, 0)$. Second, we set $\sigma_{i,j} = B[\theta(in+j) : \theta(in+j+1)]$, where $B[a : b]$ denotes bit index a to b , including a and excluding b . The same sampling routine is used for $\mathbf{s}'$ with seed $\text{seed}_{\mathbf{s}'}$.

Rounding and dividing. As usual, integers modulo a power of two r are represented in the interval $[-r/2, r/2 - 1]$. For any power-of-two divisor $d \mid r$ and any $x \in [-r/2, r/2 - 1]$, we can uniquely write $x = x_0 + x_1 d \bmod r$, where $x_0 \in [-d/2, d/2 - 1]$ and $x_1 \in [-r/(2d), r/(2d) - 1]$. This can be computed as $x_0 = x \bmod d$ and $x_1 = (x - x_0)/d \bmod r/d$. $\mathbf{b} = \lfloor \frac{\alpha}{\beta} \mathbf{w} \rfloor$ returns the rounded ring element, corresponding to $\mathbf{b} = (\mathbf{w} + \frac{\beta}{2\alpha}) \gg (\log_2 \beta - \log_2 \alpha)$, where $\gg$ denotes a right shift and each operation is applied coefficient-wise.

We shift the rounding errors by constant factors to ensure a mean zero error and to simplify the security analysis. We refer to the PhD thesis of D’Anvers for more details [D’V21, Section 3.2].

Ciphertext compression. To reduce the size of the ciphertext, we apply the lossy function $\lfloor \frac{t}{p} \cdot \rfloor$, where the value of t depends on the parameter set. While this decreases the ciphertext size, it increases the failure probability. We refer to Section 5.2.1 for a more detailed analysis of our failure probability. We do not apply compression to the public key.

Keys, ciphertexts, and messages encoding. We serialize keys, ciphertexts, and messages 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 $(t_i)_{0 \leq i < N}$, define

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

and

$$\text{Pack}((t_i)_{0 \leq i < N}, b) = \text{Enc}_b(t_0) \parallel \dots \parallel \text{Enc}_b(t_{N-1})$$

to be its serialization, where concatenation is defined such that t_0 is stored at the least significant position and $\text{Enc}_b(t_0) \bmod 2$ 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

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

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

- We define the serialization of the PKE public key pk_{pke} as

$$\text{pkEncode}(\text{seed}_{\mathbf{A}}, \mathbf{b}) = \text{seed}_{\mathbf{A}} \parallel \text{Pack}(\mathbf{b}, \epsilon_p).$$

The corresponding deserialization function is $\text{pkDecode}(pk_{\text{pke}})$. The total size of the PKE public key is $64 + \epsilon_p kn/8$ bytes.

- We define the serialization of the PKE secret key sk_{pke} as

$$\text{skEncode}(\mathbf{s}) = \text{Pack}(\mathbf{s}, \log_2(2\eta)).$$

The corresponding deserialization function is $\text{skDecode}(sk_{\text{pke}})$. The total size of the PKE secret key is $\log_2(2\eta)kn/8$ bytes.

- We define the serialization of the PKE ciphertext ct_{pke} as

$$\text{ctEncode}(c_m, \mathbf{b}') = \text{Pack}((\zeta_{0,i})_{0 \leq i < n}, \epsilon_t) \parallel \cdots \parallel \text{Pack}((\zeta_{\ell-1,i})_{0 \leq i < n}, \epsilon_t) \parallel \text{Pack}(\mathbf{b}', \epsilon_p),$$

where $c_m = (\zeta_0, \dots, \zeta_{\ell-1}) \in \mathbf{R}_{t,n}^\ell$. The corresponding deserialization function is $\text{ctDecode}(ct_{\text{pke}})$. The total size of the PKE ciphertext is $\epsilon_t \ell n/8 + \epsilon_p kn/8$ bytes.

- We define the serialization of the PKE message m as

$$\text{mEncode}(m) = \text{Pack}((\mu_{0,i})_{0 \leq i < n}, 1) \parallel \cdots \parallel \text{Pack}((\mu_{\ell-1,i})_{0 \leq i < n}, 1),$$

where $m = (\mu_0, \dots, \mu_{\ell-1}) \in \mathbf{R}_{2,n}^\ell$. Since $b = 1$, decoding returns centered representatives in $\{-1, 0\}$, which are interpreted modulo 2 as elements of $\mathbf{R}_{2,n}$. The corresponding deserialization function is $\text{mDecode}(m)$. The total size of the PKE message is $\ell n/8$ bytes.

- We define the serialization of the KEM public key to be identical to the serialization of the PKE public key.
- We define the serialization of the KEM secret key sk_{KEM} as $sk_{\text{pke}} \parallel pk_{\text{pke}} \parallel hpk \parallel z$.
- We define the serialization of the KEM ciphertext to be identical to the serialization of the PKE ciphertext: $ct_{\text{KEM}} = ct_{\text{pke}}$.
- The KEM shared secret is serialized separately as ss ; it is an output of Encaps and is not transmitted as part of the KEM ciphertext.

Fujisaki–Okamoto transformation. Our construction applies the Hofheinz–Hövelmanns–Kiltz modular Fujisaki–Okamoto transform with implicit rejection U_m^ℓ [HHK17] to the deterministic re-encryption form of Mithril.PKE . In this transform, the ciphertext is excluded from the encapsulation-time derivation of the candidate shared secret. We refer to Section 5.1.2 for the IND-CCA security discussion and the quoted QROM bounds.

5 Security Statements and Analyses

5.1 Theoretical Security

The main hard problem underlying Mithril is Ring-LWR over a ring $\mathbf{R}_q$ which we recall in Definition 2. In case of Mithril (and Octarine) the ring $\mathbf{R}_q$ is instantiated with the radical ring $\mathbf{S}_{q,n,k}$, and we refer to this instance as the RR-LWR problem. The theoretical security discussion is mostly independent from the ring itself, so we organize it in the following way:

1. We reduce the security of our PKE scheme to a similar PKE design based on Ring-LWR. Specifically, we show that the security of $\text{Mithril} := \text{Mithril}_\ell$ reduces to the security of Mithril_k , while the failure probability is no larger than that of Mithril_k , where k is the degree of the radical ring extension. See Theorem 3.
2. We appeal to standard reductions and analyses relating Mithril_k to Ring-LWR, and we instantiate a standard Fujisaki–Okamoto-style transform for the KEM. See Section 5.1.1 and Section 5.1.2.
3. Finally, we show that the radical ring choice $\mathbf{R}_q := \mathbf{S}_{q,n,k}$ is a secure Ring-LWR instance in Section 5.1.3.

Definition 2 (Decision Ring-LWR). *Let $q > p \geq 2$, let $\mathbf{R}_\nu := \mathbf{Z}_\nu[x]/(f(x))$ be rings where $\nu \in \{p, q\}$, and let $D_{\mathbf{s}}$ be a secret distribution over R_q . For a fixed secret $\mathbf{s} \in \mathbf{R}_q$ sampled from $D_{\mathbf{s}}$, the decision Ring-LWR problem is to distinguish the following two distributions:*

Real : $(\mathbf{A}_i, \lfloor \frac{p}{q} \mathbf{A}_i \mathbf{s} \rfloor)$, where $\mathbf{A}_i \leftarrow \mathbf{R}_q$ is uniformly random for $i = 1, \dots, N_{\text{samp}}$;
Rand : $(\mathbf{A}_i, \mathbf{b}_i)$, where $\mathbf{A}_i \leftarrow \mathbf{R}_q, \mathbf{b}_i \leftarrow \mathbf{R}_p$ are uniformly random for $i = 1, \dots, N_{\text{samp}}$.

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

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

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

5.1.1 IND-CPA security of Mithril.PKE

We define the IND-CPA experiment in the standard sense. The adversary receives a public key pk , outputs two equal-length messages $m_0, m_1 \in \mathbf{R}_{2,n}^\ell$, receives an encryption of m_b for a hidden bit b , and wins if it guesses b . The advantage is denoted by

$$\text{Adv}_{\text{Mithril}_\ell.\text{PKE}}^{\text{IND-CPA}}(\mathcal{A}) = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

We begin by showing that the security of Mithril_ℓ reduces to the security of Mithril_k .

Theorem 3. *For any adversary $\mathcal{A}$ running the IND-CPA experiment for Mithril_ℓ there exists an adversary $\mathcal{B}$ running the IND-CPA experiment for Mithril_k such that*

$$\text{Adv}_{\text{Mithril}_\ell.\text{PKE}}^{\text{IND-CPA}}(\mathcal{A}) = \text{Adv}_{\text{Mithril}_k.\text{PKE}}^{\text{IND-CPA}}(\mathcal{B}).$$

Moreover, writing δ_ℓ for the failure probability of Mithril_ℓ , we have $\delta_\ell \leq \delta_k$.

Proof. Let $pk = (\text{seed}_{\mathbf{A}}, \mathbf{b})$ be the public key generated by the challenger of $\mathcal{B}$. The adversary $\mathcal{B}$ forwards this public key to $\mathcal{A}$. $\mathcal{A}$ returns challenge messages m_0 and m_1 , which are (ℓn) -bit messages. $\mathcal{B}$ embeds m_0 and m_1 into the top ℓn bits of larger (kn) -bit messages $\mu_0 = 0^{(k-\ell)n} \parallel m_0$

and $\mu_1 = 0^{(k-\ell)n} \parallel m_1$ and sends them to its challenger. The challenger returns a ciphertext $c = (c_\mu, \mathbf{b}')$ which $\mathcal{B}$ converts to $c^* = (c_\mu^{[k-\ell:k-1]}, \mathbf{b}')$, which is a valid ciphertext for Mithril_ℓ . Finally $\mathcal{A}$ returns the bit b' upon receiving c^* , which $\mathcal{B}$ responds to its challenger. As c^* corresponds to the encryption of m_b with Mithril_ℓ and c corresponds to the encryption of μ_b with Mithril_k , the advantages of $\mathcal{A}$ and $\mathcal{B}$ are exactly the same.

Since $\ell \leq k$ by the design of the PKE, the ciphertext of Mithril_ℓ is a subset of the ciphertext of Mithril_k . Therefore if decryption fails for Mithril_ℓ , it also fails for Mithril_k , and it follows that $\delta_\ell \leq \delta_k$. $\square$

As Mithril_k is a regular Ring-LWR PKE, we can apply the standard reduction to the Ring Learning With Rounding problem. We refer to the thesis of D’Anvers [D’V21, Theorem 2-3] for an overview of reductions and analyses for Ring-LWR based schemes in the context of Saber. While D’Anvers treats the case of Module-LWR, we can recover Ring-LWR by setting $\ell = 1$. Following this line of analysis, the IND-CPA security of Mithril can be related to $\text{Adv}_{q,p,\mathcal{U}_2}^{\text{R-LWR}}$. This PKE reduction is not a random-oracle-model statement; random oracles enter later in the FO transform used for the KEM. The rounding and correctness estimates use the scheme’s message modulus t , and the resulting constraints are part of the concrete parameter analysis. In particular, all parameter sets satisfy the rounding-separation condition

$$\frac{q}{p} \leq \frac{p}{t}.$$

We use this as the basis for our parameter selection, as was done by Saber.

5.1.2 IND-CCA security of Mithril.KEM

We use the standard IND-CCA KEM experiment for Mithril_ℓ . Our construction follows the modular FO-with-implicit-rejection transform U_m^χ of Hofheinz–Hövelmanns–Kiltz [HHK17] applied to the deterministic re-encryption form of Mithril.PKE . For this transform we quote the standard QROM bounds from the literature. The QROM treatment of U_m^χ was first established by Hofheinz–Hövelmanns–Kiltz [HHK17], refined by Bindel et al. [BBH⁺19] and Don–Fehr–Majenz–Schaffner [DFMS19, DFMS21] via the measure-and-reprogram framework, and further tightened by Kuchta et al. [KSS⁺20] using measure-rewind-measure.

Applying these results to our PKE yields, under quantum query bounds $q_{\mathcal{G}}$ and $q_{\mathcal{H}}$, a bound of the form

$$\text{Adv}_{\text{Mithril.KEM}}^{\text{IND-CCA,QROM}}(\mathcal{A}) \leq C \cdot q_{\mathcal{G}} \cdot \sqrt{\text{Adv}_{\text{Mithril.PKE}}^{\text{OW-CPA,QROM}}(\mathcal{B})} + \mathcal{O}\left(\frac{q_{\mathcal{G}}^2 + q_{\mathcal{H}}^2}{2^{\ell n}}\right) + q_{\text{dec}} \delta_{\text{dec}} + \Delta_{\text{coll}}^q,$$

for an absolute constant C ; coarser bounds with a multiplicative factor $q_{\mathcal{G}}^2$ and no square root are obtainable directly from [HHK17].

5.1.3 Security of the RR-LWR Instance

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 Mithril .

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 Learning With Errors, 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 ring

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

is simply a concrete coefficient representation of $\mathcal{O}_{\mathbf{K}}/q\mathcal{O}_{\mathbf{K}}$. The following theorem states the representation reduction formally.

Theorem 4 (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 4 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. The relation to the canonical-embedding version of Ring-LWR is captured by the linear basis change L discussed below: its singular values $\sigma_{\min}(L) = \sqrt{nk}$ and $\sigma_{\max}(L) \leq \beta\sqrt{nk}$ quantify the distortion introduced by moving between the transported monomial basis and the canonical embedding. Our concrete attack estimates account for this transported-basis distortion using the constants β tabulated below.

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 power-of-two 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 $\beta\sqrt{nk}$ depending on k : for $k = 2$, $\beta < 1.74$, for $k = 3$, $\beta < 2.08$, for $k = 5$, $\beta < 2.41$, for $k = 7$, $\beta < 2.57$ and for $k = 15$, $\beta < 2.79$. 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 Mithril, 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 g and h 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 $S_{q,n,k}$.

5.1.4 Concrete Attack Estimate Methodology

In assessing the security of Mithril, we adopt the widely-used *Core-SVP hardness* methodology [ADPS16] as a first-order metric. This approach provides a scheme-agnostic method for comparing parameter strengths across lattice-based cryptosystems. The Core-SVP metric essentially measures how hard it is to solve the underlying lattice problem (e.g., the unique-SVP instance derived from the RR-LWR problem) at the chosen parameters, under the assumption that the best attacks use lattice reduction (e.g., the Block Korkine-Zolotarev (BKZ) algorithm [SE94, CN11]) with an optimal SVP oracle in block dimension β . By abstracting away lower-order factors (such as the polynomial-time overhead of lattice reduction and memory costs), one estimates the attack cost as roughly exponential in the BKZ block size β . We assume that a single call to an ideal length- β SVP solver costs about $2^{0.292\beta}$ basic operations in a classical setting, or $2^{0.265\beta}$ operations for a quantum attacker (accounting for the best-known Grover-like speedups for sieving-based SVP algorithms [Laa15, LMvdP15]). This coarse metric is convenient because it depends primarily on the lattice dimension and noise rate, not on the specific algebraic structure of the scheme, allowing direct comparison between different lattice-based proposals. Moreover, Core-SVP hardness tends to err on the conservative side, encouraging parameter choices that withstand attacks even under optimistic cost models for the adversary.

Scheme	Type	C	Q	β	Refined attack	fail prob.
Mithril-128	primal	134	121	458	153	-603
	dual-hybrid	129	117	443		
Mithril-256	primal	270	245	925	277	-280
	dual-hybrid	257	233	879		
Mithril-512	primal	559	507	1913	539	-230
	dual-hybrid	526	478	1802		
ML-KEM-512	primal	119	108	406	140	-139
	dual-hybrid	115	105	395		
ML-KEM-768	primal	182	165	624	196	-165
	dual-hybrid	174	158	597		
ML-KEM-1024	primal	255	232	874	262	-175
	dual-hybrid	241	219	826		
LightSaber	primal	118	107	405	140	-120
	dual-hybrid	122	111	419		
Saber	primal	190	172	649	205	-136
	dual-hybrid	182	165	624		
FireSaber	primal	260	237	891	267	-165
	dual-hybrid	248	225	848		

Table 3: Security estimates, refined attack costs and failure probability comparison between Mithril and related lattice-based schemes.

For schemes based on the **RR-LWR** problem, which in turn can be reduced to the **LWE** problem, the two main classes of lattice attacks are the *primal* and the *dual-hybrid* attack. The primal attack reduces the **LWE** instance to a unique-SVP (uSVP) instance and attempts to recover the secret directly via lattice reduction. The dual-hybrid attack, on the other hand, combines a dual lattice distinguisher with a combinatorial guessing step over part of the secret, leading to a hybrid strategy. In the **Core-SVP** framework, both attacks are typically evaluated by determining the **BKZ** block size β required to make the corresponding distinguishing or decoding step succeed, and translating this into an estimated cost.

While the **Core-SVP** metric provides a simple and robust baseline, it intentionally abstracts away many aspects of the actual attack complexity. In practice, we complement these estimates with refined cost evaluations obtained with the lattice-estimator framework of Albrecht et al. [APS15]. In this framework, the function `estimate.rough` closely corresponds to the **Core-SVP** methodology, as it derives the security level primarily from the **BKZ** block size β . This aligns with the discussion above and provides a consistent, scheme-agnostic reference point.

To obtain more realistic estimates, we additionally use the full `estimate` function, which refines the attack cost by explicitly accounting for the number of ring operations (“rop”) required by the complete attack pipeline. This includes not only the asymptotic cost of the underlying SVP oracle, but also the overhead of **BKZ** reduction, dimensions-for-free techniques, hybrid strategies, and other algorithmic optimizations. As a result, the refined estimates capture effects that are ignored in the **Core-SVP** framework, such as the cost of repeated SVP calls, non-negligible polynomial factors, and practical trade-offs between time and memory.

From a conceptual perspective, these refined estimates provide a more accurate view of concrete security, especially for classical attacks, where recent progress in lattice reduction and sieving has significantly improved practical performance. However, this increased realism comes at the price of additional modeling assumptions: the results depend on heuristics about the behavior of **BKZ**, the effectiveness of sieving implementations, and the validity of cost models for low-level operations. Consequently, refined estimates are inherently less stable over time and

more sensitive to future algorithmic improvements.

We therefore report both metrics: the Core-SVP hardness as a conservative and comparable baseline, and the refined rop-based estimates as an indication of the current best understanding of concrete attack costs. This dual view reflects the standard practice in the analysis of lattice-based schemes, where coarse but robust metrics are complemented by more detailed, yet assumption-dependent, evaluations.

Table 2 lists the security level for the different parameter sets of Mithril. Table 3 compares the security with ML-KEM, a learning with errors based KEM, and Saber, a learning with rounding based KEM. Details on how to obtain these numbers can be found in Appendix A.

5.2 Practical Security

5.2.1 Decryption failure attacks

As with most LWE/LWR-based schemes, decryption is not always perfectly correct. A decryption failure occurs when the deterministic rounding errors, combined with the interaction between the secret key and ciphertext vectors, accumulate sufficiently to shift a decrypted coefficient across the decoding boundary. Such an error can be detected and may leak information about the secret key.

A decryption failure attack actively exploits this leakage. An adversary intentionally crafts specific ciphertexts designed to artificially inflate the error term [DGJ⁺19, DRV20, DB22]. By observing whether the decryption operation succeeds or fails, the attacker learns precise linear inequalities regarding the coefficients of the secret key. By systematically accumulating enough of these failure-inducing ciphertexts, the attacker builds a system of inequalities that can be solved using statistical techniques or lattice reduction algorithms to completely recover the private key.

The decryption failures for Mithril arise in a similar fashion as other Ring- or Module-LWR schemes, such as Saber. We use a Saber-style heuristic estimator for the failure probability, following the calculation framework of [DKR⁺20, Theorem 6.2]. Translated to our notation, the estimator is based on the event

$$\delta = \Pr[\|(\mathbf{b}\mathbf{s}')^{[k-\ell:k-1]} - (\mathbf{b}'\mathbf{s})^{[k-\ell:k-1]} + \mathbf{e}_r, \text{ mod } p\|_\infty > p/4]$$

where $\mathbf{e}_r$ is represented in $\mathbf{R}_p^\ell$ with centered coefficients in $[-p/(2t), p/(2t) - 1]$. As was done previously for Saber-style estimates, we heuristically model the failure events for the shared secret bits as independent; we do not claim a strict proof of all coefficient correlations.

As we operate in a radical *ring*, it should be noted that $(\mathbf{b}\mathbf{s}')^{[k-\ell:k-1]}$ and $(\mathbf{b}'\mathbf{s})^{[k-\ell:k-1]}$ are elements of $\mathbf{R}_q^\ell$, whereas the analogous Module-LWR expression typically has elements of $\mathbf{R}_q$. In other words, secret key bits are derived from coefficients of ℓ polynomials of degree n each, as opposed to a single polynomial of n coefficients for Module-LWR. Since we select $n = 128$ instead of $n = 256$, this gives a lower failure probability for the 128-bit security level where $\ell = 1$.

However, the radical ring structure includes $k - 1 - i$ multiplications by $y + 2$ for the inner product of row i of $\mathbf{b}$ and $\mathbf{s}'$, see Equation (6). This increases the norm of its coefficients, for which we include a worst-case factor 3 in the calculation of the failure probability. As we compute the *bottom* rows we minimize the impact of the multiplication by $y + 2$. For example, for $\ell = 1$ in case of Mithril-128 only a single row is used, so no multiplications by $y + 2$ actually take place as it is not involved in the bottom row, making the multiplication equivalent to a regular inner product in case of Module-LWR.

The script given in Appendix A.2 implements this heuristic estimator for the decryption failure probability for the different parameter sets of ARCANE-Mithril listed in Table 2 and Table 3. Since the estimated logarithms of the decryption failure probabilities are well below what can practically occur, decryption failure attacks are estimated to have practically insignificant impact on the security of Mithril.

5.2.2 Side-channel attacks

The algorithms are constant-time as they do not have any secret-dependent branching. Furthermore, the use of power-of-two moduli simplifies certain computations, such as avoiding non-constant-time rejection sampling during the generation of random elements. Hence, we expect correct implementations of ARCANE-Mithril to be constant-time given that the underlying hardware instructions are also constant-time.

However, an unprotected implementation of Mithril may leak during decapsulation through power consumption or electromagnetic emanations [RRCB20, XPR⁺22, RCDB24]. Such leakage can help an adversary guess the long-term secret key. Hence, countermeasures against known side-channel attacks are essential. The deterministic execution of Mithril’s computations and the LWR foundation with power-of-two moduli enable efficient implementations of masking countermeasures. Countermeasure techniques from the Saber KEM literature [BDK⁺21] can be applied to Mithril.

- **Power-of-two moduli simplify masking.** Arithmetic masking, modular reduction, rounding, compression, and Boolean-to-arithmetic mask conversions can be implemented with low overhead using shifts and masks.
- **Deterministic LWR rounding avoids explicit error sampling.** This reduces the amount of randomness that must be masked and makes rounding easier to protect than LWE-style noise addition.
- **Fixed-time sampling without rejection.** Random elements can be generated with a fixed number of bits per coefficient, avoiding rejection loops that could complicate constant-time and masked implementations.
- **Simple secret distribution supports masked multiplication.** The small signed coefficient set $\{-2, -1, 0, 1\}$ simplifies masked handling of secret coefficients and reduces the complexity of protected arithmetic [ABC⁺23].

6 Performance Evaluation

The arithmetic design of Mithril is intentionally algorithm-agnostic with respect to the polynomial multiplication algorithm used. The specification defines the required radical-ring operations, but does not rely on a particular algorithm or element representation used for the negacyclic polynomial products. This allows implementations to choose the multiplication strategy best suited to their platform or optimization goal. To demonstrate this flexibility, our reference implementation uses Toom–Cook multiplication [Too63, Co066], whereas the AVX2 implementation uses Number-Theoretic Transforms [CT65, Pol71] over an auxiliary prime.

6.1 Arithmetic implementations

Reference implementation. The reference implementation stores coefficients and accumulates final results in 16 bits over the power-of-two coefficient ring. Multiplication in the degree-128 negacyclic sub-ring is implemented by a four-way Toom–Cook multiplication. For Mithril-512, Toom–Cook uses 32-bit evaluations and 64-bit exact products and interpolation before folding into 16-bit accumulators. The relation $x^k = y + 2$ is incorporated through a precomputed multiplication window: each requested coefficient in the radical basis is obtained as an inner product of length k over the negacyclic sub-ring. The implementation can therefore compute either the full radical-ring product or only its ℓ highest-degree coefficients. Rounding and reduction to the power-of-two moduli are fused with the final accumulation and serialization whenever possible.

AVX2 implementation. The AVX2 implementation uses the 26-bit auxiliary prime

$$53\,501\,952 + 1 = 2^{13} \cdot 3 \cdot 7 \cdot 311 + 1$$

for Mithril-128 and Mithril-256, and the 28-bit prime 268 432 897 for Mithril-512, in the length-128 Number-Theoretic Transform (NTT). Coefficients are represented in the Montgomery domain, and the public operand is transformed once and prepared for all radical-basis inner products required by the operation. For public-key and ciphertext components that are only needed through the message-bearing radical coefficients, the preparation is restricted to the ℓ required output coefficients. Forward transforms, pointwise products, accumulation, inverse transforms, and the final power-of-two rounding or reduction are vectorized with AVX2. This auxiliary prime is only an implementation-specific choice: it is chosen large enough to contain the unreduced intermediate products and does not modify any Mithril parameter or distribution.

6.2 Performance Analyses

Sampling and expansion. All three parameter sets use $n = 128$ and two-bit secret coefficients, with $\epsilon_q = 13, 13, 14$, respectively; they differ in the radical extension degree $k \in \{5, 9, 18\}$, the message dimension $\ell \in \{1, 2, 4\}$, and the ciphertext-compression modulus t . The seeded samplers produce bit-packed XOF output that is unpacked directly, without rejection. Expanding $\mathbf{A}$ requires

$$\frac{\epsilon_q n k}{8}$$

bytes of XOF output: this means 1 040, 1 872, and 4 032 bytes for Mithril-128, Mithril-256, and Mithril-512, respectively. Sampling one secret ring element requires

$$\frac{2nk}{8} = 32k$$

bytes: namely 160, 288, and 576 bytes. The sampling cost is therefore determined primarily by XOF throughput, bit unpacking, and, in the AVX2 implementation, preparation of the transform-domain representation.

Radical-ring products. For a full product, each of the k output coefficients is an inner product of length k , giving

$$N_{\text{full}} = k^2$$

base negacyclic polynomial products. Computing only the ℓ message-bearing coefficients requires

$$N_{\text{sel}} = k\ell$$

base products. Thus, for Mithril-128, Mithril-256, and Mithril-512, the full products use 25, 81, and 324 base products, whereas the polynomial products in the shared secret computation (which only need to compute part of the output, see Section 4, paragraph “Message Space”) use 5, 18, and 72 products. Together with the cost of each base product, these counts explain the principal scaling across the parameter sets.

Operation costs. Mithril.PKE key generation expands $\mathbf{A}$, samples $\mathbf{s}$, computes the full product $\mathbf{A}\mathbf{s}$, rounds from q to p , and serializes the key pair. Encryption expands $\mathbf{A}$, samples $\mathbf{s}'$, computes the full product $\mathbf{A}\mathbf{s}'$, and then computes only the ℓ highest-degree coefficients of $\mathbf{b}\mathbf{s}'$ before message addition and ciphertext compression. Decryption does not expand $\mathbf{A}$ and requires only the corresponding ℓ -coefficient product $\mathbf{b}'\mathbf{s}$, followed by message recovery.

The KEM operations additionally include the Fujisaki–Okamoto hashing and randomness generation specified in Section 4. Decapsulation performs PKE decryption, derives the candidate

key and encryption seed, deterministically re-encrypts the recovered message, derives the fallback key from the ciphertext and z , and selects the output in constant time. Because decapsulation contains both a PKE decryption and a deterministic re-encryption, its cost is comparable to the other KEM operations at the smallest parameter set and becomes the largest KEM cost for the two larger parameter sets.

The communication and storage costs are given in Table 2. The KEM public key is the PKE public key, the KEM ciphertext is the PKE ciphertext, and the KEM private key contains the PKE private key, the PKE public key, its digest, and the fallback secret z .

6.3 Performance Test

Experimental setup. All measurements were collected on an Intel Core i7-10700K CPU with a nominal frequency of 3.80 GHz, with AVX2, BMI2, and POPCNT available, running Linux 6.8.0-138-generic. The compiler was GCC 11.4.0. The reference implementation is written in C, while the AVX2 implementation combines C with x86-64 assembly. The reference implementation was compiled with `-O3`. The AVX2 implementation was compiled with `-O3, -fomit-frame-pointer, -mavx2, -mbmi2, -mpopcnt, -march=native, and -mtune=native`. In the `pseudoXOF`-based submission measurements reported in Tables 4 and 5, the benchmarked binaries instantiate the XOF and hash functions with the `pseudoXOF`-based construction in the submission package.

Each benchmark binary internally executed 1000 measured repetitions per operation following 10 warmup calls and was run serially on CPU 2 using `taskset -c 2`. Timing used the processor’s performance monitoring unit (PMU) to count CPU cycles in user and kernel mode, and the measured counter-read overhead was subtracted. The tables report medians in PMU cycles; lower values are faster. Whole-function measurements include sampling, hashing, arithmetic, packing, and all other work performed by the corresponding algorithm invocation. CPU-frequency governor and turbo-boost settings were left unchanged. For the ML-KEM comparison reported below, the ML-KEM implementation was taken from the local source tree at Git revision 3edd5af59919.

End-to-end performance. Tables 4 and 5 report the end-to-end PKE and KEM operation costs for the `pseudoXOF`-based reference and AVX2 implementations in the submission package.

Parameter set	Implementation	KeyGen	Enc	Dec
Mithril-128	Reference	261 229	285 001.5	29 459.5
	Optimized	130 985	133 616	5 259
Mithril-256	Reference	639 487.5	733 868	98 042.5
	Optimized	236 764.5	242 918.5	10 031
Mithril-512	Reference	5 028 955.5	6 075 621	1 044 501.5
	Optimized	480 143	493 836	22 930.5

Table 4: End-to-end `pseudoXOF`-based performance of Mithril.PKE in the reference and optimized implementations. Values are median PMU cycles.

The AVX2 speedup increases with k , but the end-to-end gains are smaller than the arithmetic-only gains because `pseudoXOF`-based sampling and hashing remain a substantial part of the measured path. For PKE key generation, the speedup is $1.99\times$, $2.70\times$, and $10.47\times$ for the three parameter sets; for PKE encryption it is $2.13\times$, $3.02\times$, and $12.30\times$. PKE decryption gains $5.60\times$, $9.77\times$, and $45.55\times$, because the arithmetic dominates more strongly. Both implementations compute only the ℓ selected coefficients. The KEM speedups range from $1.76\times$

Parameter set	Implementation	KeyGen	Encaps	Decaps
Mithril-128	Reference	302 651	324 044	351 341
	Optimized	171 523	171 332	172 333
Mithril-256	Reference	698 618.5	790 580.5	888 295.5
	Optimized	294 582.5	297 396	305 749
Mithril-512	Reference	5 232 025	6 277 211.5	7 364 776
	Optimized	659 789.5	679 568	715 160

Table 5: End-to-end **pseudoXOF**-based performance of Mithril.KEM in the reference and optimized implementations. Values are median PMU cycles.

to $10.30\times$, with the smaller factors caused by the additional **pseudoXOF**-based hashing in key generation, encapsulation, and decapsulation.

Arithmetic breakdown. Table 6 isolates the dominant arithmetic costs in the reference implementation.

Parameter set	Base Toom–Cook	Full product	One selected coefficient	PKE Dec
Mithril-128	4 990	123 136	24 632	29 459.5
Mithril-256	4 966.5	396 510	44 083	98 042.5
Mithril-512	14 137.5	4 527 987	250 985.5	1 044 501.5

Table 6: Selected arithmetic costs in the reference implementation. The one-coefficient column measures one highest-degree radical coefficient; PKE Dec includes all ℓ required coefficients and message recovery. Values are median PMU cycles.

The degree-128 Toom–Cook multiplication has nearly the same cost for Mithril-128 and Mithril-256, while Mithril-512 uses wider exact intermediate arithmetic and has a higher base-product cost. The full-product and one-coefficient measurements reflect the k^2 and k counts, respectively, together with this base-product cost. The end-to-end decryption path requires $k\ell$ base products, together with unpacking, reduction, rounding, and message serialization.

Table 7 gives the corresponding microbenchmarks for the AVX2 implementation.

Parameter set	Expand and prepare A	Sample secret	Transform secret	Full product and round	ℓ -coefficient product and reduce
Mithril-128	108 869.5	15 505	1 878	5 214	2 594
Mithril-256	195 859.5	27 673	3 350	10 955.5	5 087
Mithril-512	391 975	54 940	7 126	27 581	11 595

Table 7: Selected sampling and arithmetic costs in the AVX2 implementation. The two product columns are end-to-end kernels and already include the secret transform; the columns are therefore not additive. Values are median PMU cycles.

Expanding and preparing **A** and transforming the secret grow primarily with k , while the full-product kernel reflects the k^2 radical-basis structure after transform-domain preparation. The selected-coefficient product grows more slowly because it computes only ℓ output coefficients. In particular, the transform-domain representation is prepared once per operand and reused across all inner products in the same operation. The product microbenchmarks use prepared public operands.

Comparison with ML-KEM. Table 8 compares the SHAKE-based AVX2 Mithril implementation from the additional implementation directory with ML-KEM-512 AVX2 and ML-KEM-1024 AVX2 on the same platform. The Mithril-128 parameter set is compared with ML-KEM-512, and Mithril-256 is compared with ML-KEM-1024. The Mithril-512 parameter set is omitted because ML-KEM has no parameter set targeting the corresponding 512-bit classical security level. These are end-to-end comparisons: each measurement includes the hash suite, serialization, and randomness generation within the measured algorithm call.

Operation	Mithril-128 AVX2	ML-KEM-512 AVX2	Mithril-256 AVX2	ML-KEM-1024 AVX2
PKE KeyGen	20 555	15 100	35 013	39 798
PKE Enc	23 765	16 142	41 709	41 259
PKE Dec	5 220	1 240	9 999	2 133
KEM KeyGen	36 371	25 987.5	57 685	59 232
KEM Encaps	37 904	27 458	62 997	60 411
KEM Decaps	42 075	29 421.5	73 418	65 375

Table 8: End-to-end SHAKE-based AVX2 comparison with ML-KEM AVX2. Values are median PMU cycles.

The ML-KEM measurements are included as an external calibration point against a mature and highly optimized lattice KEM implementation, rather than as a strict equal-security or like-for-like comparison. The compared parameter sets have different security margins, different encoded data sizes, and rely on different algebraic assumptions. In the first AVX2 calibration point, ML-KEM-512 has lower cycle-count medians than Mithril-128 in the six measured operations. This is consistent with the fact that Mithril-128 targets a stronger estimated lattice-security margin than ML-KEM-512: our estimates give Mithril-128 classical and quantum Core-SVP costs of 129 and 117 bits, respectively, with a refined gate-count estimate of 153 bits, compared with 115, 105, and 140 bits for ML-KEM-512. At the ML-KEM-1024-level calibration point, the AVX2 results are more operation-dependent. The Mithril-256 implementation is faster in PKE key generation by approximately 12.0%, while the KEM key-generation medians differ by less than 2.7%, indicating effectively comparable performance for this operation. ML-KEM-1024 has lower medians in PKE encryption, PKE decryption, KEM encapsulation, and KEM decapsulation; however, this comparison should again be read in light of the different security margins, since Mithril-256 has estimated classical and quantum Core-SVP costs of 257 and 233 bits, with a refined estimate of 277 bits, compared with 241, 219, and 262 bits for ML-KEM-1024. These differences are also consistent with the distinct arithmetic structures of the schemes. ML-KEM uses Module-LWE over a fixed degree-(256) cyclotomic ring with a small module rank, whereas Mithril uses RR-LWR arithmetic over a radical-ring representation with $n = 128$, $k \in \{5, 9, 18\}$, and message extraction controlled by ℓ . The selected-coefficient optimization in the AVX2 implementation reduces the amount of radical-ring output required in decryption and decapsulation, but it does not eliminate the expansion, transform, and preparation costs of the backend. The measured performance should therefore be viewed as a practical baseline for the current AVX2 implementation under stronger RR-LWR parameter margins, rather than as a direct equal-cost comparison with ML-KEM.

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 analysis, and Haochen Huang to the software implementation.

Appendix

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](#).

In order to report Core-SVP-comparable security figures (as commonly used in the lattice-cryptography literature), we use the estimator’s “rough” interface `LWE.estimate.rough`. For more refined metrics we use the `LWE.estimate` call.

The classical and quantum core-SVP bit-security figures as well as the refined attacks as reported in Table 2 and Table 3 can be reproduced by running the following script in the lattice-estimator repository. In order to obtain the security estimates for Mithril-512 one needs to increase the upper bound on the blocksize by adjusting this value in `estimator/conf.py` to

```

1 max_beta = 3500

from estimator import *
2 from estimator.nd import UniformMod
3 from estimator.lwe_parameters import LWEParameters
4
5
6 def print_core_svp_table(results):
7     schemes = list(results.keys())
8
9     rows = [("bits", [f"{results[s]['bits']:.1f}" for s in schemes]),
10            ("beta", [str(results[s]["beta"]) for s in schemes]),
11            ("classical core-SVP", [f"{results[s]['classical_svp']:.1f}"
12            " for s in schemes]),
13            ("quantum core-SVP", [f"{results[s]['quantum_svp']:.1f}"
14            " for s in schemes]),
15            ("beta-extended", [str(results[s]["beta_ext"]) for s in
16            schemes]),
17            ("rop-extended", [str(results[s]["bits_ext"]) for s in
18            schemes]), ]
19
20     header = [""] + schemes
21     table = [header] + [[name] + vals for name, vals in rows]
22
23     col_widths = [max(len(str(cell)) for cell in col)
24                   for col in zip(*table)]
25
26     for row in table:
27         print(" ".join(str(cell).rjust(w)
28                         for cell, w in zip(row, col_widths)))
29
30 def best_attack(estimate):
31     return min(((results["beta"], N(log(results["rop"], 2)))
32                for attack, results in estimate.items()
33                if "beta" in results),
34               key=lambda x: x[1])

```

```

31
32 # Define our parameter sets
33 rrlwr5_params = LWEParameters(
34     n=5*128,
35     q=2**13,
36     Xs=UniformMod(4),
37     Xe=UniformMod(2**(13-11)), # p = 2^11
38     m=5*128,
39     tag="RR-LWR-5"
40 )
41 rrlwr9_params = LWEParameters(
42     n=9*128,
43     q=2**13,
44     Xs=UniformMod(4),
45     Xe=UniformMod(2**(13-11)), # p = 2^11
46     m=9*128,
47     tag="RR-LWR-9"
48 )
49 rrlwr17_params = LWEParameters(
50     n=17*128,
51     q=2**13,
52     Xs=UniformMod(4),
53     Xe=UniformMod(2**(13-11)), # p = 2^11
54     m=17*128,
55     tag="RR-LWR-17"
56 )
57
58 params = {
59     # The proposed RR-LWR parameter sets
60     "rrlwr5": rrlwr5_params,
61     "rrlwr9": rrlwr9_params,
62     "rrlwr17": rrlwr17_params,
63     # Kyber* for comparison
64     "kyber512": schemes.Kyber512,
65     "kyber768": schemes.Kyber768,
66     "kyber1024": schemes.Kyber1024,
67     # *Saber for comparison
68     "lightsaber": schemes.LightSaber,
69     "saber": schemes.Saber,
70     "firesaber": schemes.FireSaber,
71 }
72
73 results = {}
74 for name, p in params.items():
75     estimate = LWE.estimate.rough(p, quiet=True, jobs=4)
76     beta, bits = best_attack(estimate)
77     results[name] = {"beta": beta, "bits": bits }
78     estimate_extended = LWE.estimate(p, quiet=True, jobs=4)
79     beta_ext, bits_ext = best_attack(estimate_extended)
80     results[name] = {"beta": beta, "bits": bits,
81                     "beta_ext": beta_ext, "bits_ext": bits_ext }
82
83 for r in results.values():
84     beta = r["beta"]
85     r["classical_svp"] = 0.292 * beta
86     r["quantum_svp"] = 0.265 * beta
87
88 print_core_svp_table(results)

```

A.2 Failure Probability

We use the failure probability scripts from the SABER implementation at <https://github.com/KULeuven-COSIC/SABER>, commit [f7f39e4](https://github.com/KULeuven-COSIC/SABER/commit/f7f39e4), to estimate the failure probability. We make some adjustments to account for the radical ring structure, see Section 5.2.1. The failure probability numbers reported in Table 2 and Table 3 can be reproduced by running the script below.

```
1      #
2      # This file has been adapted from the implementation
3      # (available at, Public Domain https://github.com/KULeuven-COSIC/SABER/blob/master/Scripts/Python\_Scripts/select\_params/proba\_util.py)
4      #
5
6      from math import factorial as fac
7      from math import log, ceil, erf, sqrt, floor
8      import decimal
9
10     decimal.getcontext().prec = 1000
11
12     def mod_switch(x, q, rq):
13         """ Modulus switching (rounding to a different discretization
14             of the Torus)
15         :param x: value to round (integer)
16         :param q: input modulus (integer)
17         :param rq: output modulus (integer)
18         """
19         z = int(floor(1.* rq * x / q + rq/(2*q)) % rq) # Switched from
20             'round' to 'floor' to round 0.5 upwards
21
22         if (z >= rq/2):
23             z -= rq
24
25         return z
26
27     def mod_centered(x, q):
28         """ reduction mod q, centered (ie represented in -q/2 .. q/2)
29         :param x: value to round (integer)
30         :param q: input modulus (integer)
31         """
32         a = x % q
33         if a < q/2:
34             return a
35         return a - q
36
37     def build_mod_switching_error_law(q, rq):
38         """ Construct Error law: law of the difference introduced by
39             switching from and back a uniform value mod q
40         :param q: original modulus (integer)
41         :param rq: intermediate modulus (integer)
42         """
43         D = {}
44         V = {}
45         for x in range(-int(q/2), int(q/2)):
46             y = mod_switch(x, q, rq)
47             z = mod_switch(y, rq, q)
48             d = mod_centered(x - z, q)
```

```

47         D[d] = D.get(d, 0) + 1./q
48         V[y] = V.get(y, 0) + 1
49
50     return D
51
52 def law_convolution(A, B):
53     """ Construct the convolution of two laws (sum of independent
54         variables from two input laws)
55     :param A: first input law (dictionnary)
56     :param B: second input law (dictionnary)
57     """
58     C = {}
59     for a in A:
60         for b in B:
61             c = a+b
62             C[c] = C.get(c, 0) + A[a] * B[b]
63     return C
64
65 def law_convolution_neg(A, B):
66     """ Construct the convolution of two laws (sum of independent
67         variables from two input laws)
68     :param A: first input law (dictionnary)
69     :param B: second input law (dictionnary)
70     """
71     C = {}
72     for a in A:
73         for b in B:
74             c = a-b
75             C[c] = C.get(c, 0) + A[a] * B[b]
76     return C
77
78 def law_product(A, B):
79     """ Construct the law of the product of independent variables
80         from two input laws
81     :param A: first input law (dictionnary)
82     :param B: second input law (dictionnary)
83     """
84     C = {}
85     for a in A:
86         for b in B:
87             c = a*b
88             C[c] = C.get(c, 0) + A[a] * B[b]
89     return C
90
91 dist_rate = -100000 # Add this to stop pruning too early
92 def clean_dist(A):
93     """ Clean a distribution to accelerate further computation (
94         drop element of the support with proba less than 2^dist_rate
95     )
96     :param A: input law (dictionnary)
97     """
98     B = {}
99     for (x, y) in A.items():
100         if y>2**(dist_rate):
101             B[x] = y
102     return B

```

```

100
101 def iter_law_convolution(A, i):
102     """ compute the -ith fold convolution of a distribution (using
103         double-and-add)
104     :param A: first input law (dictionnary)
105     :param i: (integer)
106     """
107     D = {0: 1.0}
108     i_bin = bin(i)[2:] # binary representation of n
109     for ch in i_bin:
110         D = law_convolution(D, D)
111         D = clean_dist(D)
112         if ch == '1':
113             D = law_convolution(D, A)
114             D = clean_dist(D)
115     return D
116
117 def convolution_remove_dependency(A, B, q, p):
118     normalizer={}
119     maxa=int(q/p)
120     for a in A:
121         normalizer[a%maxa]=normalizer.get(a%maxa,0)+A[a]
122
123     C={}
124     for a in A:
125         for b in B:
126             c=a-b
127             if (c%maxa==0):
128                 C[c] = C.get(c, 0) + A[a] * B[b]/normalizer[a%maxa]
129     return C
130
131 def tail_probability(D, t):
132     """
133     Probability that an drawn from D is strictly greater than t in
134     absolute value
135     :param D: Law (Dictionnary)
136     :param t: tail parameter (integer)
137     """
138     s = 0
139     ma = max(D.keys())
140     if t >= ma:
141         return 0
142     for i in reversed(range(int(ceil(t)), ma)): # Summing in
143         reverse for better numerical precision (assuming tails are
144         decreasing)
145         s += D.get(i, 0) + D.get(-i, 0)
146     return s
147
148 def build_uniform_law(k):
149     """ Construct the binomial law as a dictionnary
150     :param k: (integer)
151     :param x: (integer)
152     :returns: A dictionnary {x:p_k(x) for x in {-k..k}}
153     """
154     D = {}
155     for i in range(-k, k):
156         D[i] = 1/(2*k)
157     return D

```

```

154
155 def get_failure_prob(D_s, q, p, T, k, ell, n):
156
157     # Compute rounding error distribution
158     D_e = build_mod_switching_error_law(q, p)
159
160     assert D_e == D_s # True for Mithril parameter sets
161
162     # Compute shared secret error distribution
163     se = law_product(D_s, D_e) # Dist on single product of input
        polynomial coefficients
164
165     prob_secret_success_polys = ell*[0]
166     for i in range(ell):
167
168         # For row i we get k-i polynomials without factor (y+2), i
        polynomials with (y+2) that get an additional factor 3
169         # Total (k-i) + 3*i = k+2*i
170         se2 = iter_law_convolution(se, (k+2*i)*n) # Dist on
        accumulated polynomial coefficient of s'*e, assume each
        coefficient is independent
171         se2 = law_convolution_neg(se2, se2) # Dist on accumulated
        polynomial coefficient of s'*e - e'*s
172         e2 = build_mod_switching_error_law(q, T) # Dist on
        ciphertext error
173         D = law_convolution(se2, e2) # Dist in shared secret
        coefficient
174
175         prob_coeff_fail = tail_probability(D, q/4.) # Failure
        probability of a single shared secret bit
176         a = decimal.Decimal(prob_coeff_fail)
177         prob_secret_success = (1-a)**n # Success probability for
        all bits in this polynomial
178         prob_secret_success_polys[i] = prob_secret_success
179
180     # Compute success probability across all secret key polynomials
181     prob_success = decimal.Decimal(1)
182     for i in range(ell):
183         prob_success *= prob_secret_success_polys[i]
184
185     prob_fail = log(1-prob_success, 2)
186
187     return prob_fail
188
189 def main():
190     D_s = build_uniform_law(2)
191     prob128 = get_failure_prob(D_s, 2**13, 2**11, 2**3, 5, 1, 128)
192     print('Failure probability 128-bit security level: ', prob128)
193     prob256 = get_failure_prob(D_s, 2**13, 2**11, 2**3, 9, 2, 128)
194     print('Failure probability 256-bit security level: ', prob256)
195     prob512 = get_failure_prob(D_s, 2**13, 2**11, 2**8, 17, 4, 128)
196     print('Failure probability 512-bit security level: ', prob512)
197
198     main()

```

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 1.
- **PKE algorithms KeyGen, Encrypt, and Decrypt:** Section 4, Algorithm 1.
- **KEM algorithms KeyGen, Encaps, and Decaps:** Section 4, Algorithm 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 Mithril-128, Mithril-256, and Mithril-512:** Section 4, Table 2.
- **Key, ciphertext, and shared-secret sizes:** Section 4, Table 2.
- **Failure-probability 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 2.2, Table 1; Section 4, Table 2; Section 5, Table 3; and Section 6, Tables 4–8.
- **Algorithm specification:** Section 4, Algorithm 1.

- **2. Design Rationale**

- **Main design ideas and rationale:** Sections 1 and 2.1.
- **Mathematical hard problem:** Sections 2.1, 3.4, and 5.1.3.
- **RR-LWR as the underlying assumption:** Sections 1, 2.1, 3.4, and 5.1.3.
- **Criterion for parameter selection:** Sections 1 and 2.1, and Table 2.
- **Choice of radical-ring structure:** Sections 1, 2.1, and 3.4.
- **Choice of n , k , p , q , t , and ℓ :** Section 2.1 and Table 2.
- **Use of power-of-two moduli and uniform sampling:** Sections 1 and 2.1.
- **Detailed analysis of failure probability and its impact on security:** Sections 2.1 and 5.2.1, and Appendix A.2.
- **Other design considerations:** Section 2.1, including Sampling, Partial ring elements, Fujisaki–Okamoto transformations, and High-assurance implementation.
- **Relation to existing work and comparative rationale:** Section 2.2.

- **3. Security Statements and Analyses**

- **Security statements of the algorithm:** Section 5.
- **Theoretical security:** Section 5.1.
- **RR-LWR hard problem:** Section 5.1.3.

- **IND-CPA security of Mithril.PKE:** Section 5.1.1, Theorem 3.
 - **IND-CCA security of Mithril.KEM:** Section 5.1.2.
 - **QROM bound discussion for Mithril.KEM:** Section 5.1.2.
 - **Security model, reductions, and bounds:** Section 5.1.1, Theorem 3, and Section 5.1.2.
 - **Quantum-model security analysis:** Section 5.1.
 - **Practical security methodology:** Section 5.1.4.
 - **Complexity of known classical and quantum attacks for each instance:** Section 5.1.4, Table 3; Section 4, Table 2; and Appendix A.1.
 - **Reproducible security-estimation script:** Appendix A.1.
 - **Decryption-failure attacks:** Section 5.2.1.
 - **Side-channel attacks and constant-time / masking considerations:** Section 5.2.2.
- **4. Performance Evaluation**
 - **Performance analyses:** Sections 6.1 and 6.2.
 - **Arithmetic implementation analysis:** Section 6.1.
 - **Reference implementation arithmetic:** Section 6.1.
 - **AVX2 implementation arithmetic:** Section 6.1.
 - **Sampling and expansion cost analysis:** Section 6.2.
 - **Radical-ring product cost analysis:** Section 6.2.
 - **Operation-level performance analysis:** Section 6.2.
 - **Communication and storage cost analysis:** Section 6.2 and Table 2.
 - **Performance test results for all algorithm instances on popular 64-bit PC processors:** Section 6.3.
 - **Detailed configuration of implementation platforms:** Section 6.3, Experimental setup. The document specifies the CPU, clock rate, instruction sets, operating system, compiler, implementation language, and benchmark method.
 - **PKE performance results:** Section 6.3, Table 4.
 - **KEM performance results:** Section 6.3, Table 5.
 - **Arithmetic breakdown results:** Section 6.3, Tables 6 and 7.
 - **Comparison with Kyber AVX2 implementation:** Section 6.3, Table 8.
 - **5. Feature Statements**
 - **Feature statements, including innovativeness, simplicity, flexibility, compatibility, extensibility, and performance advantages on variant platforms:** Section 1.

References

- [ABC⁺23] Aikata, Andrea Basso, Gaëtan Cassiers, Ahmet Can Mert, and Sujoy Sinha Roy. Kavach: Lightweight masking techniques for polynomial arithmetic in lattice-based cryptography. *IACR TCHES*, 2023(3):366–390, 2023. [doi:10.46586/tches.v2023.i3.366-390](https://doi.org/10.46586/tches.v2023.i3.366-390).

- [ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In *Proceedings of the 25th USENIX Security Symposium*, pages 327–343. USENIX Association, 2016.
- [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).
- [BDK⁺21] Michiel Van Beirendonck, Jan-Pieter D’anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *J. Emerg. Technol. Comput. Syst.*, 17(2), 2021. [doi:10.1145/3429983](https://doi.org/10.1145/3429983).
- [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).
- [BHH⁺19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random

- oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 61–90. Springer, Cham, December 2019. doi:[10.1007/978-3-030-36033-7_3](https://doi.org/10.1007/978-3-030-36033-7_3).
- [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>.
- [CCHY24] Jung Hee Cheon, Hyeongmin Choe, Dongyeon Hong, and MinJune Yi. SMAUG: Pushing lattice-based key encapsulation mechanisms to the limits. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 127–146. Springer, Cham, August 2024. doi:[10.1007/978-3-031-53368-6_7](https://doi.org/10.1007/978-3-031-53368-6_7).
- [CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. 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](https://doi.org/10.1007/978-3-662-44709-3_11).
- [CHK⁺21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for NTT-unfriendly rings. *IACR TCHES*, 2021(2):159–188, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8791>, doi:[10.46586/tches.v2021.i2.159-188](https://doi.org/10.46586/tches.v2021.i2.159-188).
- [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](https://doi.org/10.1007/3-540-48405-1_26).
- [CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, December 2011. doi:[10.1007/978-3-642-25385-0_1](https://doi.org/10.1007/978-3-642-25385-0_1).
- [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.

- [DB22] Jan-Pieter D’Anvers and Senne Batsleer. Multitarget decryption failure attacks and their application to Saber and Kyber. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 3–33. Springer, Cham, March 2022. doi:10.1007/978-3-030-97121-2_1.
- [DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Cham, August 2019. doi:10.1007/978-3-030-26951-7_13.
- [DFMS21] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. Cryptology ePrint Archive, Report 2021/280, 2021. URL: <https://eprint.iacr.org/2021/280>.
- [DGJ⁺19] Jan-Pieter D’Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 565–598. Springer, Cham, April 2019. doi:10.1007/978-3-030-17259-6_19.
- [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>.
- [DKRV18] Jan-Pieter D’Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Cham, May 2018. doi:10.1007/978-3-319-89339-6_16.
- [DRV20] Jan-Pieter D’Anvers, Mélissa Rossi, and Fernando Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 3–33. Springer, Cham, May 2020. doi:10.1007/978-3-030-45727-3_1.
- [DVV21] Jan-Pieter D’Anvers, Ingrid Verbauwhede, and Frédéric Vercauteren. Design and security analysis of lattice-based post-quantum encryption, 2021.
- [FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_34.

- [FVR⁺22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR TCHES*, 2022(1):414–460, 2022. doi:[10.46586/tches.v2022.i1.414-460](https://doi.org/10.46586/tches.v2022.i1.414-460).
- [HCCSS24] Jung Hee Cheon, Hyeongmin Choe, Jungjoo Seo, and Hyoeun Seong. SMAUG(-T), revisited: Timing-secure, more compact, less failure. *IEEE Access*, 12:188386–188397, 2024. doi:[10.1109/ACCESS.2024.3511346](https://doi.org/10.1109/ACCESS.2024.3511346).
- [HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017. doi:[10.1007/978-3-319-70500-2_12](https://doi.org/10.1007/978-3-319-70500-2_12).
- [KDvB⁺22] Suparna Kundu, Jan-Pieter D’Anvers, Michiel van Beirendonck, Angshuman Karmakar, and Ingrid Verbauwhede. Higher-order masked Saber. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 93–116. Springer, Cham, September 2022. doi:[10.1007/978-3-031-14791-3_5](https://doi.org/10.1007/978-3-031-14791-3_5).
- [KKV23] Suparna Kundu, Angshuman Karmakar, and Ingrid Verbauwhede. On the masking-friendly designs for post-quantum cryptography. In Francesco Regazzoni, Bodhisatwa Mazumdar, and Sri Parameswaran, editors, *Security, Privacy, and Applied Cryptography Engineering - 13th International Conference, SPACE 2023, Roorkee, India, December 14-17, 2023, Proceedings*, volume 14412 of *Lecture Notes in Computer Science*, pages 162–184. Springer, 2023. doi:[10.1007/978-3-031-51583-5_10](https://doi.org/10.1007/978-3-031-51583-5_10).
- [KNK⁺25] Suparna Kundu, Quinten Norga, Angshuman Karmakar, Shreya Gangopadhyay, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. Scabbard: An exploratory study on hardware aware design choices of learning with rounding-based key encapsulation mechanisms. *ACM Trans. Embed. Comput. Syst.*, 24(1):10:1–10:40, 2025. doi:[10.1145/3696208](https://doi.org/10.1145/3696208).
- [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.
- [KSS⁺20] Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shifeng Sun. Measure-rewind-measure: Tighter quantum random oracle model proofs for one-way to hiding and CCA security. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 703–728. Springer, Cham, May 2020. doi:[10.1007/978-3-030-45727-3_24](https://doi.org/10.1007/978-3-030-45727-3_24).
- [Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 3–22. Springer, Berlin, Heidelberg, August 2015. doi:[10.1007/978-3-662-47989-6_1](https://doi.org/10.1007/978-3-662-47989-6_1).
- [LMvdP15] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *DCC*, 77(2-3):375–400, 2015. doi:[10.1007/s10623-015-0067-5](https://doi.org/10.1007/s10623-015-0067-5).

- [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](https://doi.org/10.1007/978-3-642-13190-5_1).
- [LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. doi:[10.1007/s10623-014-9938-4](https://doi.org/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](https://doi.org/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](https://doi.org/10.1007/978-3-642-29011-4_43).
- [MKKV21] Jose Maria Bermudo Mera, Angshuman Karmakar, Suparna Kundu, and Ingrid Verbauwhede. Scabbard: a suite of efficient learning with rounding key-encapsulation mechanisms. *IACR TCHES*, 2021(4):474–509, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/9073>, doi:[10.46586/tches.v2021.i4.474-509](https://doi.org/10.46586/tches.v2021.i4.474-509).
- [MTK⁺20] Jose Maria Bermudo Mera, Furkan Turan, Angshuman Karmakar, Sujoy Sinha Roy, and Ingrid Verbauwhede. Compact domain-specific co-processor for accelerating module lattice-based KEM. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020. doi:[10.1109/DAC18072.2020.9218727](https://doi.org/10.1109/DAC18072.2020.9218727).
- [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](https://doi.org/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).
- [PJP⁺22] Seunghwan Park, Chi-Gon Jung, Aesun Park, Joongeun Choi, and Honggoo Kang. TiGER: Tiny bandwidth key encapsulation mechanism for easy migration based on RLWE(R). Cryptology ePrint Archive, Report 2022/1651, 2022. URL: <https://eprint.iacr.org/2022/1651>.
- [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.
- [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.
- [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.
- [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.
- [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.
- [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.
- [RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based PKE and kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020. URL: <https://doi.org/10.13154/tches.v2020.i3.307-335>, doi:10.13154/TCHES.V2020.I3.307-335.
- [SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. doi:10.1007/BF01581144.
- [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.
- [XPR⁺22] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David Oswald, Wang Yao, and Zhiming Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber. *IEEE Transactions on Computers*, 71(9):2163–2176, 2022. doi:10.1109/TC.2021.3122997.