IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2026, No. 2, pp. 372–408. DOI:10.46586/tches.v2026.i2.372-408

Lightweight PQ KEM and Hybrid MQTT Protocol for 8-bit AVR Sensor Nodes

Yifan Dong[1*], YoungBeom Kim[2,3*], Jieyu Zheng[1✉], Zhichuang Liang[1], Boyue Fang[1], Seog Chung Seo[2,3], Maire O'Neill[3] and Yunlei Zhao[1]

[1] College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China
{yfdong24,jyzheng23,zcliang21}@m.fudan.edu.cn,{byfang16,ylzhao}@fudan.edu.cn
[2] Kookmin University, Seoul, Republic of Korea
darania@kookmin.ac.kr,scseo@kookmin.ac.kr
[3] Queen's University Belfast, Northern Ireland, United Kingdom
m.oneill@ecit.qub.ac.uk

Abstract.
Most PQC schemes remain too resource-intensive for ultra-constrained 8-bit AVR wireless sensor nodes. In this work, we present a comprehensive approach to practical lightweight PQC for such devices, covering scheme design, implementation optimization, and protocol integration. Our contributions are threefold: (i) We propose CTRU-Light, a lattice-based KEM specifically tailored for IoT sensor nodes. It combines small moduli, low-degree polynomials, and NTT-friendly arithmetic for high efficiency, with ASCON used for lightweight symmetric operations. (ii) We explore NTT-friendly moduli for the first time to accelerate modular multiplication on 8-bit AVR platforms and design optimized variants of Montgomery and Barrett multiplication. We show that K-RED2X multiplication exhibits approximate equivalence to Montgomery multiplication under small NTT-friendly moduli. We apply these optimizations to the latest implementations of Kyber (ASIACCS 2025) and Saber (CHES 2025), achieving significant improvements in both speed and code size. Furthermore, we present a highly optimized AVR assembly implementation of CTRU-Light that delivers high efficiency and low stack usage. (iii) We design a Hybrid KEM–MQTT protocol that integrates classical ECDH with post-quantum KEMs. We present the first implementation of this protocol and provide a detailed empirical analysis of its performance. Experiments show that CTRU-Light is the only scheme capable of supporting both pure PQ and hybrid KEM–MQTT on 8-bit WSNs, achieving lower handshake latency than Kyber-512 and LightSaber.

Keywords: Post-quantum cryptography · Lightweight cryptography · 8-bit AVR · Modular multiplication · Lightweight protocol

1 Introduction

The emergence of quantum computing poses an existential threat to widely deployed public-key cryptosystems, including RSA and ECC. To address this, NIST has standardized several post-quantum cryptographic (PQC) algorithms, such as ML-KEM [NIS24a] and signature schemes ML-DSA [NIS24b], SLH-DSA [NIS24c], and FN-DSA [NIS24d]. However, these standardized PQC schemes impose heavy computational, memory, and bandwidth requirements, posing significant challenges for resource-constrained environments.

In parallel with the PQC standardization effort, NIST has also launched the Lightweight Cryptography (LWC) project [TTM+23]. NIST LWC standardization targets symmetric

*These authors contributed equally to this work.

Licensed under Creative Commons License CC-BY 4.0.

cryptographic primitives, and considers metrics like code size, speed, and RAM usage on extremely resource-constrained platforms such as 8-bit AVR. However, PQC introduces challenges that go far beyond those in symmetric cryptography. As Saarinen highlighted at the Second NIST PQC Standardization Conference [Saa19], adopting PQC requires not only the heavy use of symmetric primitives like AES, GCM, SHA, and SHAKE, but also additional operations such as large polynomial multiplications and matrix–vector computations. These requirements push constrained devices far beyond their resource limits.

Wireless Sensor Networks (WSNs) are distributed systems composed of numerous small, low-power sensor nodes, such as 8-bit AVR microcontrollers (e.g., ATmega4808) that feature only a few kilobytes of RAM and flash memory. Despite these limitations, WSNs are extensively used in various domains. It's also critical to ensure confidentiality and authenticity in WSNs.

1.1 Motivation

Platform-Level Gap: Understudied yet Widely Deployed 8-bit AVR Nodes. 8-bit AVR microcontrollers remain indispensable in practice, powering ultra-low-cost sensor nodes and duty-cycled endpoints in large-scale WSN deployments. Although NIST selected the ARM Cortex-M4 as the reference embedded platform for post-quantum standardization, real-world IoT networks continue to depend heavily on AVR-based devices. As emphasized by Atkins [Der21] at the third PQC standardization conference, *"while the PQM4 is focused on the M4 processor (as is NIST), we propose an assumption about porting to smaller devices, and encourage NIST to consider devices smaller than the M4 when evaluating PQC methods."* Despite this recognition, only limited work has explored AVR architectures. Existing studies are largely confined to partial implementation analyses [HKS25] or simplified simulations [KS25, Der21], offering neither comprehensive nor memory-efficient realizations. This disconnect between the widespread industrial adoption of 8-bit platforms and their limited academic coverage highlights an urgent need for systematic investigation of post-quantum security on AVR-based WSNs.

Scheme-Level Gap: Lightweight KEM Design. Most NIST PQC schemes employ large moduli and high-dimensional lattices, making their deployment on highly constrained devices particularly challenging. Implementations of these schemes often require aggressive stack management to fit within the limited on-chip memory of embedded platforms [BBB+24b, BBC+23, HHL+23, BRS22]. Such optimizations, however, usually come at the expense of execution speed, and in many cases, even highly tuned implementations still fail to run on severely constrained devices [KS25]. In contrast, lightweight PQC schemes that are natively designed with memory and stack limitations in mind can achieve far better efficiency without relying on extreme optimization techniques. This underscores the importance of developing algorithms that are intrinsically lightweight, enabling practical post-quantum security in resource-limited IoT environments. Existing studies on 8-bit AVR platforms primarily focus on adapting standard NIST candidates [HKS25, KS25, CGRR21, CGRR19, CGRR20]. Some lightweight proposals, such as Rudraksh [KGK+25], have been evaluated only on hardware accelerators, leaving their end-to-end feasibility on low-cost 8-bit sensor nodes largely unexplored. Consequently, a clear design–implementation gap persists between lightweight KEM theory and its practical realization in real-world IoT and WSN deployments.

Implementation-Level Gap: Modular Multiplication on 8-bit AVR Platforms. Modular multiplication is inherently one of the most time-consuming operations in lattice-based cryptography, as it dominates the arithmetic complexity of polynomial and matrix oper-

ations. This bottleneck becomes even more pronounced on 8-bit AVR microcontrollers, which lack hardware support for long integer multiplication and are constrained by small register widths. Previous studies on ARM Cortex M4 [AHKS22, HZZ+22, GKS21, ABC+20] primarily focus on optimization strategies that leverage specialized hardware instructions. However, these techniques are not directly applicable to 8-bit AVR devices. Existing AVR implementations often employ LUT-based multiplication [SLP+18, KS25] or Barrett reduction [HKS25], both of which incur extra flash consumption due to precomputed tables or constants, leading to larger code size. Moreover, prior work largely overlooks register utilization, which is critical for balancing performance, code size, and maintainability. Techniques that jointly minimize register pressure, flash usage, and cycle counts are still largely unexplored, suggesting room for further optimization. Addressing this gap requires modular multiplication techniques specifically designed for the architectural constraints of 8-bit AVRs, paving the way for efficient lattice-based cryptography on 8-bit AVR platforms.

Protocol-Level Gap: Hybrid KEM-MQTT on 8-bit Platforms. Ensuring post-quantum security in MQTT-based WSNs introduces further challenges. MQTT itself lacks built-in cryptographic protection, and post-quantum signatures are too heavy for sensor nodes. Hybrid approaches that combine PQC and classical algorithms (e.g., ECDH) are crucial to ensure security continuity during the transition to the post-quantum era, as they provide resistance against both classical and quantum adversaries, though at the cost of increased bandwidth and memory consumption. Recent studies [EI25, AAK24, KS25] have explored hybrid TLS 1.3 and KEM-MQTT protocols, yet such efforts remain limited to ARM or desktop platforms. Notably, [EI25] also identifies investigating the applicability of hybrid KEMTLS in constrained environments such as IoT and 5G networks as future work. The feasibility of deploying hybrid post-quantum key exchange on 8-bit AVRs therefore remains unclear.

1.2 Contributions

Building on the above motivation, our contributions are threefold:

- **Lightweight KEM Design:** We present CTRU-Light, a lattice-based post-quantum KEM for 8-bit AVR sensor nodes. We combine the structural simplicity of the NTRU problem with parameter choices featuring small moduli, low dimensions, and NTT-friendliness. CTRU-Light minimizes the complexity of polynomial operations inherent in lattice-based PQC design and ensures compactness favorable for deployment on extremely constrained devices. Furthermore, we replace the widely used heavy SHA-3 family with the NIST-standardized LWC ASCON to instantiate the symmetric primitives in CTRU-Light. Our scheme achieves IND-CCA security and provides a security strength roughly equivalent to AES-128 (corresponding to security level 1 of Kyber). Compared to both NIST-standardized PQC KEMs and existing lightweight proposals, CTRU-Light offers significantly reduced communication overhead and improved implementation efficiency.

- **Highly Optimized Implementation and Novel Modular Multiplication:** We present a highly optimized AVR assembly implementation of CTRU-Light. Compared to the C reference implementation, our design achieves speedups of 2.44×, 1.82×, and 1.97× in KeyGen, Encaps, and Decaps, respectively. The corresponding stack usage is only 2,351 bytes, 2,463 bytes, and 3,028 bytes—well within the 6 KB SRAM available on the ATmega4808 microcontroller. Importantly, CTRU-Light achieves these results *without any stack-saving optimization that sacrifices speed*. Overall, the implementation completes in 5,337 kcycles, outperforming Kyber-512 (18,282 kcycles) and LightSaber (20,235 kcycles). We present the first systematic use of NTT-friendly

moduli to accelerate modular multiplication. Building on this property, we optimize Montgomery multiplication to achieve an efficient balance among speed, flash usage, and register pressure. For small moduli, we formally demonstrate the approximate equivalence between K-RED2X and Montgomery multiplication. This validates that small moduli yield high efficiency on 8-bit AVR and supports their use in lightweight lattice-based schemes. Using the same approach, we apply our improved modular multiplication to Kyber implementation in ASIACCS 2025, improving KeyGen, Encaps, and Decaps by 1–13% with a 72.2% reduction in code size. For Saber, the same optimization achieves 24–32% performance gains with a 14.7% reduction in code size relative to the CHES 2025 implementation. The proposed techniques are broadly applicable to NTT-friendly moduli admitting a 256-th root of unity, improving the efficiency of lattice-based cryptography on 8-bit AVR platforms.

- **System-Level Hybrid Protocol Feasibility and Empirical Analysis:** We propose Hybrid KEM–MQTT, an extension of KEM-MQTT [KS25] that integrates classical ECDH with post-quantum KEMs. The protocol preserves the certificate-less and pre-distributed key features of KEM-MQTT, while enabling flexible instantiation with different KEM algorithms. To demonstrate practicality, we implement both pure and hybrid variants on 8-bit AVR platforms using CTRU-Light, Kyber-512, and LightSaber, and replace AES-GCM with the lightweight Ascon-AEAD. Experimental results show that Hybrid KEM–MQTT introduces only a modest memory overhead, and that CTRU-Light is the only scheme fully deployable on ATmega4808. On higher-end AVR devices, CTRU-Light achieves handshake latencies of 0.53 s (pure) and 2.62 s (hybrid), outperforming Kyber-512 (1.64 s / 3.73 s) and LightSaber (2.00 s / 4.08 s). These results underscore the importance of lightweight PQC KEMs for practical hybrid post-quantum secure communication in resource-constrained WSN environments.

Code. Our code is available at https://github.com/whYBeKim/CTRU-Light-and-PQC-on-AVR.

2 Preliminaries

2.1 Notion

Let $n = 2^k$ for some integer $k > 0$, and let q be a prime, we define $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. Throughout this paper, regular font letters denote polynomials in $\mathcal{R}_q$ or integers in $\mathbb{Z}$, bold lowercase letters denote column vectors, and bold uppercase letters denote matrices. Any polynomial $f \in \mathcal{R}_q$ can be expressed as $f = \sum_{i=1}^{n-1} f_i x^i$, where $f_i \in \mathbb{Z}_q$.

For any real number x, $\lceil x \rceil$ denotes the least integer greater than or equal to x, and $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x. Moreover, $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$, i.e., rounding x to the nearest integer. For an even (resp. odd) positive integer α, we define $r' = r \bmod^{\pm} \alpha$ as the unique integer in $(-\alpha/2, \alpha/2]$ (resp. $(-(\alpha-1)/2, (\alpha-1)/2]$) such that $\alpha | (r - r')$. For any positive integer α, we define $r' = r \bmod^{+} \alpha$ as the unique integer in $[0, \alpha)$ such that $\alpha | (r - r')$. Finally, when we simply write $r' = r \bmod \alpha$, the distinction between $\bmod^{\pm}$ and $\bmod^{+}$ is understood to be irrelevant.

For any $w \in \mathbb{Z}$, we denote $\|w\|_{q,\infty} = \left| w \bmod^{\pm} q \right|$. For $f \in \mathcal{R}_q$, its infinity norm is defined as $\|f\|_{q,\infty} = \max_i \|f_i\|_{q,\infty}$. For an n-dimension vector $\mathbf{w} = (w_0, \cdots, w_{n-1}) \in \mathbb{Z}^n$, we define its Euclidean norm as $\|\mathbf{w}\|_{q,2} = \sqrt{\|w_0\|_{q,\infty}^2 + \cdots + \|w_{n-1}\|_{q,\infty}^2}$.

For a finite set S, the notation $x \leftarrow S$ denotes sampling x uniformly at random from S. For a probability distribution D, the notation $x \leftarrow D$ denotes sampling x according to D. Given a positive integer η, the centered binomial distribution $\mathcal{B}_\eta$ is defined as follows: a sample $x \leftarrow \mathcal{B}_\eta$ is given by $x = \sum_{i=1}^{\eta} (a_i - b_i)$, where $(a_1, ..., a_\eta, b_1, ..., b_\eta) \leftarrow \{0,1\}^{2\eta}$. The above sampling operations extend naturally to the case of polynomials.

2.2 Cryptographic Primitives

Definition 1 (PKE). A public-key encryption (PKE) scheme is a tuple of three algorithms (KeyGen,Enc,Dec) along with a message space $\mathcal{M}$. It enables two parties to transmit secret data through public channels.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$: Given the security parameter 1^λ, the key-generation algorithm KeyGen outputs a public-secret key pair (pk, sk).
- $ct \leftarrow \mathsf{Enc}(pk, m)$: Given a public key pk and a message $m \in \mathcal{M}$ (sometimes alongside a random coin r), the encryption algorithm Enc produces the ciphertext ct for m.
- $m \leftarrow \mathsf{Dec}(sk, ct)$: Given a secret key sk and a ciphertext ct, the deterministic decryption algorithm Dec returns a message $m \in \mathcal{M}$ or a special symbol $\perp$ denoting failure.

Correctness. Let δ be the probability of decryption failure. We say that a PKE is $(1-\delta)$-correct if $\mathbf{E}[\max_{m\in\mathcal{M}} \Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m]] < \delta$.
IND-CPA security. We say that a PKE has indistinguishable encryptions under chosen-plaintext attacks (IND-CPA secure) if for any probabilistic polynomial-time adversary $\mathcal{A}$, we have $\mathbf{Adv}^{ind-cpa}_{\mathsf{PKE}}(\mathcal{A})$ negligible. The IND-CPA-advantage of $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{ind-cpa}_{\mathsf{PKE}}(\mathcal{A}) := \left| \Pr\left[b = b' : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ (m_0, m_1, s) \leftarrow \mathcal{A}(pk); \\ b \leftarrow \{0,1\}; ct^* \leftarrow \mathsf{Enc}(pk, m_b); \\ b' \leftarrow \mathcal{A}(pk, ct^*, s) \end{array} \right] - \frac{1}{2} \right|.$$

Definition 2 (KEM). A key encapsulation mechanism (KEM) is a set of algorithms that enables two parties to securely establish a shared secret key on a public channel. A KEM = (KeyGen,Encaps,Decaps) over key space $\mathcal{K}$ consists of three algorithms.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$: Given the security parameter 1^λ, the key-generation algorithm KeyGen outputs a public-secret key pair (pk, sk).
- $(ct, K) \leftarrow \mathsf{Encaps}(pk)$: Given a public key pk, the encapsulation algorithm Encaps returns a shared key $K \in \mathcal{K}$ as well as the corresponding ciphertext ct.
- $m \leftarrow \mathsf{Decaps}(sk, ct)$: Given a secret key sk and a ciphertext ct, the deterministic decryption algorithm Decaps outputs a shared key K' or $\perp$ for explicit rejection.

Correctness. If $\Pr[\mathsf{Decaps}(sk, ct) \neq K : (ct, K) \leftarrow \mathsf{Encaps}(pk)] < \delta$, we say that a KEM is $(1-\delta)$-correct
IND-CCA security. We say that a KEM has indistinguishability under chosen-ciphertext attacks (IND-CCA secure) if for any probabilistic polynomial-time adversary $\mathcal{A}$ who has access to a Decaps oracle, we have $\mathbf{Adv}^{ind-cca}_{\mathsf{KEM}}(\mathcal{A})$ negligible. The IND-CCA-advantage of $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{ind-cca}_{\mathsf{KEM}}(\mathcal{A}) := \left| \Pr\left[b = b' : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ b \leftarrow \{0,1\}; \\ (ct^*, K_0) \leftarrow \mathsf{Encaps}(pk); K_1 \leftarrow \mathcal{K} \\ b' \leftarrow \mathcal{A}^{Decaps(\cdot)}(pk, ct^*, K_b) \end{array} \right] - \frac{1}{2} \right|.$$

where the Decaps oracle is defined as $Decaps(\cdot) := \mathsf{Decaps}(sk, \cdot)$, and the adversary $\mathcal{A}$ is not allowed to make queries with input ct^*.

2.3 Hardness Assumption

Definition 3 (NTRU Assumption [HPS98]). Let Ψ be a distribution over a polynomial ring $\mathcal{R}$, and let $p \in \mathcal{R}$ be invertible. Sample $f', g \leftarrow \Psi$ and define $f = pf' + 1$, conditioned

on $f \in \mathcal{R}$ invertible. Set $h = gf^{-1} \in \mathcal{R}$. The *decisional NTRU problem* is to distinguish h from a uniformly random element over $\mathcal{R}$. We say that the decisional NTRU problem is computationally hard if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage

$$\mathbf{Adv}^{ntru}_{\mathcal{R},\Psi}(\mathcal{A}) := \left| \Pr\left[b = 1 : \begin{array}{c} f', g \leftarrow \Psi; f = pf' + 1; \\ f^{-1} \in \mathcal{R}; h = g/f; \\ b \leftarrow \mathcal{A}(h) \end{array} \right] - \Pr\left[b' = 1 : \begin{array}{c} h \leftarrow \mathcal{R}; \\ b' \leftarrow \mathcal{A}(h) \end{array} \right] \right|.$$

is negligible.

Definition 4 (**RLWR Assumption** [BPR12])**.** Let $q > p \geq 2$ be integers, and let Ψ be a distribution over a polynomial ring $\mathcal{R}$. Define $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ and $\mathcal{R}_p = \mathcal{R}/p\mathcal{R}$ be the quotient rings. The *(decisional) Ring-Learning with Rounding (RLWR) problem* is to distinguish uniform samples $(h, c) \leftarrow \mathcal{R}_q \times \mathcal{R}_p$ from samples where $h \leftarrow \mathcal{R}_q$ and $c = \lfloor \frac{p}{q} hr \rceil \bmod p, r \leftarrow \Psi$. We say that the decisional RLWR problem is computationally hard if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, its advantage $\mathbf{Adv}^{rlwr}_{\mathcal{R},\Psi}(\mathcal{A})$ is negligible, where

$$\mathbf{Adv}^{rlwr}_{\mathcal{R},\Psi}(\mathcal{A}) := \left| \Pr\left[b = 1 : \begin{array}{c} h \leftarrow \mathcal{R}_q; r \leftarrow \Psi; \\ c = \lfloor \frac{p}{q} hr \rceil \bmod p \in \mathcal{R}_p; \\ b \leftarrow \mathcal{A}(h, c) \end{array} \right] - \Pr\left[b' = 1 : \begin{array}{c} h \leftarrow \mathcal{R}_q; \\ c \leftarrow \mathcal{R}_p; \\ b' \leftarrow \mathcal{A}(h, c) \end{array} \right] \right|.$$

2.4 Number Theory Transform.

The Number Theoretic Transform (NTT) is a widely used technique for efficient polynomial multiplication. Specifically, for polynomials $f, g \in \mathbb{Z}_q[x]$, their product can be computed as $c = a \cdot b = \mathsf{INTT}(\mathsf{NTT}(a) \odot \mathsf{NTT}(b))$, where NTT and INTT denote the forward and inverse NTT, respectively, and "$\odot$" denotes point-wise multiplication. The point-wise multiplication requires only $O(n)$ time. In this way, the overall complexity can be reduced from $O(n^2)$ to $O(n \log n)$. The radix-n FFT-Trick gives the isomorphism:

$$\mathbb{Z}_q[x]/\left(x^{nm} - \zeta^n\right) \cong \prod_{i=0}^{n-1} \mathbb{Z}_q[x]/\left(x^m - \rho^i \zeta\right),$$

where ρ is an n-th root of unity modulo q, and $\zeta \in \mathbb{Z}_q$ is invertible. When $m = 1$, the transform is called a complete (or "fully splitting") NTT; otherwise it is called incomplete. The most common NTT-friendly rings correspond to the cases $\zeta^n = 1$ or $\zeta^n = -1$, which require $n \mid (q-1)$ or $2n \mid (q-1)$, respectively, in order to admit complete NTTs.

2.5 8-bit AVR Microcontroller

The AVR microcontroller architecture, introduced by Atmel (now Microchip) in 1996 [Man16], remains a representative platform for low-power embedded systems and lightweight cryptography evaluation. Compared with ARM processors, AVR poses unique software challenges, featuring an 8-bit datapath, a limited number of registers, and a strict instruction set. Its architecture includes 32 8-bit general-purpose registers connected to the ALU, three 16-bit pointer registers (`X`, `Y`, `Z`) for indirect addressing, along with a simple instruction schedule: single-cycle arithmetic and two-cycle memory access. However, fixed register usage in several instructions (e.g., `ldi`, `mul`, `adiw`) and non-uniform addressing modes restrict compiler flexibility. In practice, cryptographic implementations on AVR demand precise register allocation, minimal load/store operations, and instruction-level optimization to achieve acceptable performance under stringent hardware constraints.

2.6 KEM-MQTT

KEM-MQTT is a lightweight protocol for WSNs on 8-bit AVR, proposed by Kim *et al.* [KS25]. After demonstrating that even highly optimized PQC-DSA (e.g., Dilithium, Falcon, and SPHINCS+) remains impractical to implement in full-stream on 8-bit AVR, they adopted KEMTLS [SSW20] methods as an alternative to PQC-DSA.

The design of KEM-MQTT closely follows that of KEMTLS-PDK [SSW21], a variant of KEMTLS that uses pre-distributed public keys. Compared to TLS 1.3, KEMTLS-PDK reduces the amount of data exchanged during the handshake. In the MQTT model, a publisher periodically sends sensor data to a central broker, which manages message routing and delivery to subscribers. Likewise, KEM-MQTT achieves explicit authentication of both the publisher and the broker without incurring additional round trips. While the TLS-PDK approach allows clients to bypass server certificate validation using cached information [ST16], it was rarely adopted in the pre-quantum setting due to the relatively small size of classical certificates.

3 CTRU-Light: A Lightweight PQC KEM

In this section, we describe CTRU-Light, a lightweight NTRU-based KEM designed for ultra-resource-constrained sensor nodes.

3.1 Design Rationale

In guiding the design of our scheme, we summarize four key features as security, efficiency, compactness, and simplicity.

Security. Our scheme is expected to achieve approximately 128-bit classical security strength and over 110-bit quantum security strength, corresponding to security level 1 in the NIST Post-Quantum Cryptography standardization, and is suitable for resource-constrained devices and lightweight protocols.

Structural Simplicity. Among the various approaches to post-quantum cryptography, lattice-based cryptography is regarded as the most promising technology due to its well-established security reductions, simple polynomial operations, and low communication overhead. For example, at NIST security level 1, the bandwidth of the lattice-based ML-KEM scheme is only one-fourth that of the code-based HQC scheme.

The two most famous cryptographic hardness assumptions for lattices are {M,R}LWE, originating from Regev *et al.* [Reg09, LPR10, LS15], and NTRU, originating from Hoffstein, Pipher, and Silverman [HPS98]. Among these, NTRU enjoys advantages over Ring-LWE and Module-LWE schemes primarily in speed and compactness. Encapsulation and decapsulation in Ring-LWE often involve costly operations such as computing a public polynomial from a seed, whereas NTRU avoids this overhead. Moreover, NTRU ciphertexts are more compact in size than Ring-LWE ciphertexts because they comprise a single ring element rather than two. This compactness is particularly beneficial in cryptographic primitives that combine encryption with zero-knowledge proofs [LS19]. For these reasons, we adopt NTRU lattices in our lightweight scheme.

Compactness. In NTRU-based KEMs, the algebraic structure of NTRU allows relatively small key sizes and ciphertexts. While {M,R}LWE schemes achieve 128-bit security in a 512-dimensional polynomial ring, NTRU generally demands a larger dimension, typically ranging from 700 to 800, for equivalent security (e.g., NTRU-HRSS [HRSS17] uses 701, and NTRU-Prime [BCLvV17] uses 761). The large polynomial degree not only escalates storage and computational costs but also complicates the selection of suitable NTT rings for acceleration. To mitigate these issues, we leverage the following techniques to reduce the scheme's ring dimension to 512 while maintaining NIST Security Level 1:

- *Small moduli.* We achieve a ring dimension reduction from ≈ 700 to 512 by selecting a 10-bit modulus q, in contrast to the 13-bit moduli of other NTRU-based schemes. This smaller q ensures the 125-bit security level while simultaneously lowering bandwidth and the memory required for polynomial calculations.

- *The usage of scalable E_8 lattice code.* The small moduli increase the decryption failure rate. To fix this, we use the scalable E_8 lattice code by Liang *et al.* [LFZZ24a]. This code is based on the extended Hamming code. The error correction guarantees an ideal decryption failure rate of $\approx 2^{-\lambda}$. (λ is the security strength.) Additionally, the scheme uses only a single ciphertext polynomial. This provides inherent ciphertext compression. The compression further reduces the overall bandwidth.

- *The usage of vector-based coding.* We extend the scalable E_8 lattice code by integrating a vector-based encoding mechanism inspired by [ZFY23, ADPS16a, PG13]. During key generation, we replace the usual integer constant with the polynomial $p = 1 - x^{\frac{n}{2}}$. The value $\frac{2}{q+1}p^{-1}$ is then used to apply a secondary encoding to the E_8 lattice-coded message. This modification aims to reduce the error contribution associated with p, compared to the constant term used in standard NTRU.

Efficiency. Consistent with the design philosophy outlined in [BGG+16], which advocates avoiding large multipliers and complex noise samplers in lightweight cryptography, our scheme is developed from an implementation-friendly standpoint.

- *NTT-friendly scheme.* In NTRU-based cryptographic schemes, polynomial multiplication and inversion over $\mathcal{R}_q$ are two major bottlenecks. Due to limitations on parameter selection, most NTRU-based KEMs are built upon polynomial rings of the form $\mathbb{Z}_q[x]/(x^p - 1)$, with a power-of-2 modulus and prime degree. However, such rings are incompatible with the use of the Number Theoretic Transform (NTT) to accelerate polynomial operations. By reducing the ring dimension to 512, we can instead employ cyclotomic rings of the form $\mathbb{Z}_q[x]/(x^n + 1)$, where q is chosen as an NTT-friendly modulus. This allows efficient use of the NTT, which is particularly advantageous for speeding up polynomial inversion in NTRU-based schemes.

- *The centered binomial secret distribution.* We sample both secret polynomials in CTRU-Light.KeyGen and CTRU-Light.Enc from a centered binomial distribution, instead of using a ternary distribution as in [HPS98, BCLvV17, ZFY23]. This choice simplifies implementation.

- *Decryption in $\mathcal{R}_q$.* In [LFZZ24a] and several other NTRU-based KEMs [FFW+25], the polynomial multiplication in the decryption procedure is performed over $\mathbb{Z}_{2^m}[x]/\Phi(x)$, since such a choice leads to a lower decryption failure rate. In contrast, we give a construction in $\mathcal{R}_q$. This design fully exploits the NTT-friendly property to achieve higher computational efficiency in resource-constrained environments, while reducing stack usage and code size.

3.2 Our Scheme: CTRU-Light

The IND-CCA secure lightweight KEM, denoted as CTRU-Light, is constructed by applying $\mathsf{FO}^{\not\perp}_{\mathsf{ID}(pk),m}$ [DHK+21] to an IND-CPA secure PKE scheme, named CTRU-Light.PKE.

Given $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ with $n = 2^k$ and q an NTT-friendly prime, and q_2 is a power-of-two modulus such that $q_2 < q$, let $\mathcal{M} = \{0,1\}^{n/4}$ denote the message space. For the instantiation with $n = 512$, we define $p = 1 - x^{256}$, then we have $\frac{2}{q+1}p^{-1} = 1 + x^{256}$. The CTRU-Light.PKE scheme is described in Algorithms 1–3.

Algorithm 1 CTRU-Light.PKE.KeyGen(1^κ)

Repeat
  $f', g \leftarrow \Psi_1$
  $f := pf' + 1$
Until f^{-1} exists in $\mathcal{R}_q$
$h := g/f$
return $(pk := h, sk := f)$

Algorithm 2 CTRU-Light.PKE.Enc ($pk = h$, $m \in \mathcal{M}$)

$r \leftarrow \Psi_2$
$\sigma := hr$
$c := \left\lfloor \frac{q_2}{q} \left(\sigma + (1 + x^{256}) \cdot \mathsf{PolyEncode}(m)\right) \right\rceil \bmod q_2$
return c

Algorithm 3 CTRU-Light.PKE.Dec($sk = f$, c)

$\tilde{m} := \lfloor \frac{q}{q_2} c \rceil \cdot f \bmod^{\pm} q$
$m := \mathsf{PolyDecode}(\tilde{m})$
return m

Algorithms 4 and 5 present the PolyEncode and PolyDecode procedures, respectively. In both algorithms, we define $\mathbf{c} = (0, 1, 0, 1, 0, 1, 0, 1)$ and $C = \{(x_1, x_1, x_2, x_2, x_3, x_3, x_4, x_4) \in \{0,1\}^8 \mid \sum x_i \equiv 0 \bmod 2\}$. Under these definitions, the notation $\lambda \cdot [C \cup (C + \mathbf{c})]$ denotes a scaled E_8 lattice for $\lambda = \frac{q}{2}$ and a scaled E_8' lattice for $\lambda = \frac{q+1}{2}$.

Algorithm 4 PolyEncode($m = \sum_{i=0}^{|m|-1} m_i x^i \in \mathcal{M}$)

$E_8 = \frac{q}{2} \cdot [C \cup (C + \mathbf{c})]$
for $i = 0, \ldots, |m|/4 - 1$ **do**
  $\mathbf{k}_i := (m_{4i}, m_{4i+1}, m_{4i+2}, m_{4i+3}) \in \{0,1\}^4$
  $(v_{8i}, v_{8i+1}, \ldots, v_{8i+7}) := \mathsf{Encode}_{E_8}(\mathbf{k}_i)$
end for
$v := \sum_{i=0}^{2|m|-1} v_i x^i$
return v

Algorithm 5 PolyDecode($v = \sum_{i=0}^{n-1} v_i x^i \in \mathcal{R}_q$)

$E_8' = \frac{q+1}{2} \cdot [C \cup (C + \mathbf{c})]$
for $i = 0, \ldots, |m|/4 - 1$ **do**
  $\mathbf{x}_i := (v_{8i}, v_{8i+1}, \ldots, v_{8i+7}) \in \mathbb{R}^8$
  $(m_{4i}, m_{4i+1}, m_{4i+2}, m_{4i+3}) := \mathsf{Decode}_{E_8'}(\mathbf{x}_i) \in \{0,1\}^4$
end for
$m := \sum_{i=0}^{|m|-1} m_i x^i \in \mathcal{M}$
return m

In PolyEncode, each message $m \in \mathcal{M}$ is divided into groups of four elements (quadruples), which are then encoded using Encode_{E_8}. In contrast, PolyDecode partitions the first half of $\tilde{m} \in \mathcal{R}_q$ into octets, which are then decoded using $\mathsf{Decode}E_8'$. For detailed descriptions of Encode_{E_8} and $\mathsf{Decode}_{E_8'}$, readers can refer to [LFZZ24a].

Let $\mathsf{G} : \{0,1\}^* \to \mathcal{K} \times \mathcal{COINS}$ and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ be hash functions, where $\mathcal{COINS}$ denotes the randomness space of CTRU-Light.PKE and $\mathcal{K}$ denotes the shared key space of CTRU-Light. Let $\mathcal{PK}$ denote the public-key space of CTRU-Light.PKE, and let $\mathsf{ID} : \mathcal{PK} \to \{0,1\}^\gamma$ be a fixed-output-length deterministic function. The IND-CCA-secure key encapsulation mechanism CTRU-Light is described in Algorithms 6 through 8.

Algorithm 6 CTRU-Light.KeyGen(1^κ)

1: $(pk, sk) \leftarrow$ CTRU-Light.PKE.KeyGen(1^κ)
2: $z \leftarrow \{0,1\}^\iota$
3: **return** $(pk' := pk, sk' := (sk, pk, z))$

Algorithm 7 CTRU-Light.Encaps(pk)

1: $m \leftarrow \mathcal{M}$
2: $(K, coin) := \mathsf{G}(\mathsf{ID}(pk), m)$
3: $c := \mathsf{CTRU\text{-}Light.PKE.Enc}(pk, m; coin)$
4: **return** (c, K)

Algorithm 8 CTRU-Light.Decaps($(sk, z), c$)

1: $m' := \mathsf{CTRU\text{-}Light.PKE.Dec}(sk, c)$
2: $(K', coin') := \mathsf{G}(\mathsf{ID}(pk), m')$
3: $\tilde{K} := \mathsf{H}(\mathsf{ID}(pk), z, c)$
4: **if** $c = \mathsf{CTRU\text{-}Light.PKE.Enc}(pk, m'; coin')$ **then**
5: **return** K'
6: **else**
7: **return** $\tilde{K}$
8: **end if**

3.3 Analysis of Decryption Failure

Theorem 1 (Correctness of CTRU-Light). *Let Ψ_1, Ψ_2 be two distributions on the ring $\mathcal{R}$, and Ψ_3, Ψ_4 be the following distributions over $\mathcal{R}$.*

- Ψ_3. *Choose uniform-random $x \leftarrow \mathcal{R}_q$, and return $\frac{q_2}{q}x - \lfloor \frac{q_2}{q}x \rceil \bmod^{\pm} q_2$.*
- Ψ_4. *Choose uniform-random $x \leftarrow \mathcal{R}_{q_2}$, and return $\frac{q}{q_2}x - \lfloor \frac{q}{q_2}x \rceil \bmod^{\pm} q$.*

We sample $f', g \leftarrow \Psi_1$ and $r, e \leftarrow \Psi_2$. Let $p = 1 - x^{256}$ and define $f = pf' + 1$. For a message $m \in \mathcal{M}$, let $\lambda \cdot s$ denote the output of PolyEncode(m)*, where λ is the scaling factor and $s \in \{0,1\}^{2|m|}$ is the corresponding binary encoding. For $1 \le i \le n/16$, let err_i denote the i-th octet of $gr - (\frac{q}{q_2}\varepsilon_1 + \varepsilon_2)f - \frac{s}{2}$, where $\varepsilon_1 \leftarrow \Psi_3, \varepsilon_2 \leftarrow \Psi_4$. Define*

$$1 - \delta = \Pr\left[\|err_i\|_{q+1,2} < \frac{q+1}{2}, 1 \le i \le n/16\right].$$

Then, the decryption failure probability of CTRU-Light *equals δ.*

Proof. Since $p = 1 - x^{256}$, we have $p^{-1} = \frac{q+1}{2}(1 + x^{256})$. Therefore, the computation of the ciphertext c in CTRU-Light can be expressed as

$$\begin{aligned} c &= \left\lfloor \frac{q_2}{q}\left(\sigma + (1 + x^{256}) \cdot \mathsf{PolyEncode}(m)\right) \right\rceil \bmod q_2 \\ &= \left\lfloor \frac{q_2}{q} hr + \frac{q_2}{q} \cdot \frac{q}{2} \cdot (1 + x^{256}) \cdot s \right\rceil \bmod q_2 \\ &= \left\lfloor \frac{q_2}{q} hr \right\rceil + \frac{q_2(1 + x^{256})}{2} s \bmod q_2, \end{aligned}$$

where $s \in \mathcal{R}_2$. Based on the hardness of the NTRU assumption, h is pseudorandom in $\mathcal{R}_q$. $\lfloor \frac{q_2}{q} hr \rceil$ represents an RLWR sample, while $\frac{q_2(1+x^{256})}{2} s = \frac{q_2}{q+1}(p^{-1}s)$ corresponds to the encoding of m.

In the decryption procedure, we have $m = \mathsf{PolyDecode}_{E_8'}\big(\lfloor \frac{q}{q_2} c \rceil \cdot f \bmod^{\pm} q\big)$. From the expression of c, the following result is derived.

$$\begin{aligned}
\lfloor \frac{q}{q_2} c \rceil \cdot f \bmod^{\pm} q &= \lfloor \frac{q}{q_2} (\lfloor \frac{q_2}{q} hr \rceil + \frac{q_2(1+x^{256})}{2} s) \rceil \cdot f \bmod^{\pm} q \\
&= (\frac{q}{q_2} (\lfloor \frac{q_2}{q} hr \rceil + \frac{q_2(1+x^{256})}{2} s) - \varepsilon_2) \cdot f \bmod^{\pm} q \\
&= (\frac{q}{q_2} (\frac{q_2}{q} hr - \varepsilon_1 + \frac{q_2(1+x^{256})}{2} s) - \varepsilon_2) \cdot f \bmod^{\pm} q \\
&= (hr - \frac{q}{q_2}\varepsilon_1 - \varepsilon_2 + \frac{q(1+x^{256})}{2} s) \cdot f \bmod^{\pm} q \\
&= gr - (\frac{q}{q_2}\varepsilon_1 + \varepsilon_2) f + \frac{q(1+x^{256})}{2} s(pf' + 1) \bmod^{\pm} q \\
&= gr - (\frac{q}{q_2}\varepsilon_1 + \varepsilon_2) f + \frac{q+1}{2}(1+x^{256}) s - \frac{(1+x^{256})s}{2} \bmod^{\pm} q,
\end{aligned}$$

where $\varepsilon_1 \leftarrow \Psi_3$ and $\varepsilon_2 \leftarrow \Psi_4$. In $\mathbb{Z}_q[x]/(x^{512}+1)$, the terms $\frac{q+1}{2}s$ and $\frac{q+1}{2}x^{256}s$ represent the first and second halves of $\frac{q+1}{2}(1+x^{256})s$, respectively. Likewise, $\frac{s}{2}$ and $\frac{x^{256}}{2}s$ represent the first and second halves of $\frac{1}{2}(1+x^{256})s$, respectively.

Since only the first half of $\lfloor \frac{q}{q_2} c \rceil \cdot f$ is decoded by $\mathsf{PolyDecode}_{E_8'}$, we only need to consider $gr - (\frac{q}{q_2}\varepsilon_1 + \varepsilon_2) f + \frac{q+1}{2} s - \frac{s}{2}$. Each octet of $\frac{q+1}{2}s$ corresponds to a lattice point in the scalable E_8' lattice with scale factor $\frac{q+1}{2}$, written as $\frac{q+1}{2}(\mathbf{k}_i \mathbf{H} \bmod 2)$. Let err_i denote the i-th octet of $gr - (\frac{q}{q_2}\varepsilon_1 + \varepsilon_2) f - \frac{s}{2}$. From [LFZZ24a], correct recovery of $\mathbf{k}_i$ requires $\|err_i\|_{q+1,2} < \frac{q+1}{2}$. □

3.4 Security Analysis

3.4.1 Provable Security

In this subsection, we establish the IND-CCA security of CTRU-Light in the random oracle model. NTRU-based PKE schemes, such as NTRU-HRSS [CDH+20] from NIST PQC Round 3, naturally achieve OW-CPA security based on the hardness of the NTRU assumption. The same property holds for our CTRU-Light.PKE. Thanks to its construction, CTRU-Light.PKE also achieves IND-CPA security. The RLWR assumption is optional in this context, as it is only used to obtain tighter reduction bounds for KEM security proofs in the quantum random oracle model.

Theorem 2 (IND-CPA security of CTRU-Light.PKE). *For any probabilistic polynomial-time adversary $\mathcal{A}$, there exist probabilistic polynomial-time adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{ind-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{A}) \leq \mathbf{Adv}^{ntru}_{\mathcal{R}_q,\Psi_1}(\mathcal{B}) + \mathbf{Adv}^{rlwr}_{\mathcal{R}_q,\Psi_2}(\mathcal{C}).$$

Proof. We finish the proof of Theorem 2 through a sequence of games $\mathbf{G}_i$. Let $\mathcal{A}$ be the adversary against the IND-CPA security of CTRU-Light.PKE. $\mathbf{Succ}_i$ denotes the event that $\mathcal{A}$ wins in the game $\mathbf{G}_i$.

Game $\mathbf{G}_0$. This game is the original IND-CPA security experiment. Therefore, we have $\mathbf{Adv}^{ind-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{A}) = |\Pr[\mathbf{Succ}_0] - 1/2|$.

Game $\mathbf{G}_1$. This game is the same as $\mathbf{G}_0$, except that replacing $h = g/f$ by $h \leftarrow \mathcal{R}_q$ in the key generation. To distinguish $\mathbf{G}_1$ from $\mathbf{G}_0$ is equivalent to solving an NTRU problem.

More precisely, there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ such that $|\Pr[\mathbf{Succ}_0] - \Pr[\mathbf{Succ}_1]| \leq \mathbf{Adv}^{ntru}_{\mathcal{R}_q,\Psi_1}(\mathcal{B})$.

Game $\mathbf{G}_2$. This game is the same as $\mathbf{G}_1$, except that $\mathbf{G}_2$ writes the ciphertext as $c = \lfloor \frac{q_2}{q} hr \rceil + \frac{q_2}{2}(1+x^{256})s$, where $s \in \mathcal{R}_2$. Then, $\mathbf{G}_2$ replace $\lfloor \frac{q_2}{q} hr \rceil$ by a uniformly random element in $\mathcal{R}_{q_2}$. Thus, there exists an adversary $\mathcal{C}$ with the same running time as that of $\mathcal{A}$ such that $|\Pr[\mathbf{Succ}_1] - \Pr[\mathbf{Succ}_2]| \leq \mathbf{Adv}^{rlwr}_{\mathcal{R}_q,\Psi_2}(\mathcal{C})$.

In Game $\mathbf{G}_2$, for any given m_b, the information of m_b in $\mathsf{PolyEncode}(m_b)$ is completely hidden by the uniformly random element from $\mathcal{R}_q$. Therefore, the ciphertext c leaks no information about m_b. Then, we have $\Pr[\mathbf{Succ}_2] = 1/2$.

Combining all the probabilities finishes the proof. □

Since CTRU-Light is constructed via the $\mathsf{FO}^{\not\perp}_{\mathsf{ID}(pk),m}$ transformation applied to the underlying PKE, its CCA security from OW-CPA- and IND-CPA-secure PKEs is established in Theorems 3 and 4, respectively, following the results of [DHK+21, HHK17, JZC+18, JZM19].

Theorem 3 (OW-CPA PKE $\Longrightarrow$ IND-CCA KEM). *For any (quantum) adversary $\mathcal{A}$ making at most q_D decapsulation queries and q_H (quantum) random oracle queries against the IND-CCA security of the KEM, there exists a (quantum) adversary $\mathcal{B}$ with roughly the same running time as $\mathcal{A}$ such that:*

- *In the random oracle model, it holds that*

$$\mathbf{Adv}^{ind-cca}_{\mathsf{CTRU\text{-}Light}}(\mathcal{A}) \leq q_H \cdot \mathbf{Adv}^{ow-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{B}) + \frac{q_H}{2^\iota} + (q_H + q_D)\delta;$$

- *In the quantum random oracle model, setting $q_{HD} := q_H + q_D$, it holds that*

$$\mathbf{Adv}^{ind-cca}_{\mathsf{CTRU\text{-}Light}}(\mathcal{A}) \leq 2q_{HD}\sqrt{\mathbf{Adv}^{ow-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{B})} + \frac{2q_H}{\sqrt{2^\iota}} + 16q^2_{HD}\delta.$$

Theorem 4 (IND-CPA PKE $\Longrightarrow$ IND-CCA KEM). *For any (quantum) adversary $\mathcal{A}$ making at most q_D decapsulation queries and q_H (quantum) random oracle queries against the IND-CCA security of the KEM, there exists a (quantum) adversary $\mathcal{B}$ with roughly the same running time as $\mathcal{A}$, such that:*

- *In the random oracle model, it holds that*

$$\mathbf{Adv}^{ind-cca}_{\mathsf{CTRU\text{-}Light}}(\mathcal{A}) \leq 2\left(\mathbf{Adv}^{ind-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{B}) + \frac{q_H}{|\mathcal{M}|}\right) + \frac{q_H}{2^\iota} + (q_H + q_D)\delta;$$

- *In the quantum random oracle model, setting $q_{HD} := q_H + q_D + 1$, it holds that*

$$\mathbf{Adv}^{ind-cca}_{\mathsf{CTRU\text{-}Light}}(\mathcal{A}) \leq 2\sqrt{q_{HD}\mathbf{Adv}^{ind-cpa}_{\mathsf{CTRU\text{-}Light.PKE}}(\mathcal{B}) + \frac{2q^2_{HD}}{|\mathcal{M}|}} + \frac{2q_H}{\sqrt{2^\iota}} + 16q^2_{HD}\delta.$$

When CTRU-Light.PKE attains IND-CPA security, the CCA reduction is tighter. In the ROM, the IND-CPA reduction is tight, while the OW-CPA reduction incurs a q_H loss. In the QROM, both reductions have quadratic loss, but the IND-CPA case yields a smaller factor $\sqrt{q_H + q_D + 1}$ compared to $q_H + q_D$ for the OW-CPA case.

3.4.2 Concrete Security

For lattice-based cryptosystems based on NTRU and RLWR, the dominant attack is the primal attack [ACD+18].

Primal Attack. The primal attack aims to solve the unique Shortest Vector Problem (u-SVP) in an integer *embedding lattice*.

- The NTRU problem can be modeled as a u-SVP instance in the NTRU lattice, generated by the basis matrix

$$\begin{bmatrix} q\mathbf{I}_n & \mathbf{M}_n(h) \\ 0 & \mathbf{I}_n \end{bmatrix} \in \mathbb{Z}^{2n\times 2n},$$

  where $\mathbf{M}_n(h)$ is the "rotation matrix" of h, whose i-th row corresponds to $x^i h \bmod (x^n+1)$. The secret pair (f, g) is the target short vector in this lattice.

- To construct a u-SVP instance from the RLWR problem, we scale the RLWR sample $\lfloor \frac{q_2}{q} hr \rceil$ into $t := \frac{q}{q_2}\lfloor \frac{q_2}{q} hr \rceil = hr - \frac{q}{q_2}\left(\frac{q_2}{q} hr - \lfloor \frac{q_2}{q} hr \rceil\right) = hr + e \bmod q$. The corresponding lattice basis is given by

$$\begin{bmatrix} q\mathbf{I}_n & -\mathbf{M}_n(h) & \mathbf{V}_n(t) \\ 0 & \mathbf{I}_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{Z}^{(2n+1)\times(2n+1)},$$

  where $\mathbf{V}_n(t)$ denotes the vector form of t, and the secret $(e, r, 1)$ is the unique short vector.

Other attacks. The dual attack [MR08] employs the BKZ algorithm in the dual lattice to solve the *decisional* RLWE problem, aiming to partially recover the secret and derive the full secret vector. However, due to its massive storage requirements, the dual attack is considered less practical than the primal attack, and is typically excluded from concrete security estimates in lattice-based cryptographic schemes, e.g., Kyber in NIST PQC Round 3 [ABD+20]. We likewise omit the dual attack for CTRU-Light. Similarly, the overstretched NTRU attacks [DvW21], which apply when the modulus is much larger than the secret polynomial coefficients, are also not relevant for CTRU-Light.

Core-SVP Methodology. Following the Core-SVP methodology [ADPS16b], we consider only the cost of a single invocation of an SVP oracle in the BKZ [CN11] algorithm to derive a conservative estimation. For a b-dimensional SVP oracle, the best known classic algorithm takes the complexity of $2^{0.292b+o(b)}$, while the quantum algorithm takes the complexity of $2^{0.265b+o(b)}$[1]. We evaluate the classical and quantum security of CTRU-Light using a Python script adapted from [ABD+20, LFZZ24b], with results shown in Table 1.

3.5 Parameter Set

Table 1 summarizes the recommended parameters of CTRU-Light. The scheme is instantiated over the power-of-two cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ with $n = 512$, targeting NIST security level 1. We set the 10-bit prime modulus $q = 769$ for efficient NTT-based operations, fix $p = 1 - x^{256}$, and choose message length $|m| = 128$. The table reports the shared key size $|K|$, public key size $|pk|$, ciphertext size $|ct|$, and bandwidth $|pk| + |ct|$ (in bytes). Security is evaluated against primal attacks, where "C" and "Q" denote estimated classical and quantum security (bits), "B. W." is the bandwidth of the scheme, and the last column "δ" gives the error probability computed via a Python script (Section 3.3).

Table 1: Recommended Parameters of CTRU-Light.

Name	n	q	q_2	(Ψ_1, Ψ_2)	$\lvert K\rvert$	$\lvert sk\rvert$	$\lvert pk\rvert$	$\lvert ct\rvert$	B.W.	(C, Q)	δ
CTRU-Light	512	769	2^8	$(\mathcal{B}_1, \mathcal{B}_1)$	16	1504	640	512	1152	(125,114)	2^{-168}

We also provide a comparison with existing post-quantum KEMs. Table 2 presents a comparison between CTRU-Light and mainstream NTRU-based KEM schemes, the NIST

[1] Recent advancements in quantum random walks [CL21] demonstrate an improved quantum complexity of $2^{0.257b+o(b)}$, though this approach has not yet been widely integrated into LWE-based security analyses.

Table 2: Comparisons between CTRU-Light and other KEMs.

Name	Assumptions	Rings	n	q	k	B.W.	δ
CTRU-Light (this work)	NTRU,RLWR	$\mathbb{Z}_q[x]/(x^n+1)$	512	769	1	1152	2^{-168}
MNTRU-I [BJL+24]	MNTRU	$\mathbb{Z}_q[x]/(x^n+1)$	256	1409	2	1372	2^{-127}
BAT-512 [FKPY22]	NTRU,RLWR	$\mathbb{Z}_q[x]/(x^n+1)$	512	257	1	994	2^{-146}
NEV-512 [ZFY23]	NTRU,RLWE	$\mathbb{Z}_q[x]/(x^n+1)$	512	769	1	1230	2^{-138}
NEV'-512 [ZFY23]	NTRU,RLWE	$\mathbb{Z}_q[x]/(x^n+1)$	512	769	1	1230	2^{-200}
CTRU-512 [LFZZ24a]	NTRU,RLWE	$\mathbb{Z}_q[x]/(x^n-x^{\frac{n}{2}}-1)$	512	3457	1	1408	2^{-143}
NTRU-HPS$^{677}_{2048}$ [CDH+19]	NTRU	$\mathbb{Z}_q[x]/(x^n-1)$	677	2048	1	1860	$2^{-\infty}$
HRSS-701 [CDH+19]	NTRU	$\mathbb{Z}_q[x]/(x^n-1)$	701	8192	1	2276	$2^{-\infty}$
NTTRU-768 [LS19]	NTRU	$\mathbb{Z}_q[x]/(x^n-x^{\frac{n}{2}}-1)$	768	7681	1	2496	2^{-1217}
SNTRUP-761 [BBC+20]	NTRU	$\mathbb{Z}_q[x]/(x^n-x-1)$	761	4591	1	2197	$2^{-\infty}$
ML-KEM-512 [ABD+20]	MLWE	$\mathbb{Z}_q[x]/(x^n+1)$	256	3329	2	1568	2^{-139}
HQC-128 [MAB+20]	QCSD	–	–	–	–	6746	$< 2^{-128}$
Rudraksh-poly64 [KGK+25]	MLWE	$\mathbb{Z}_q[x]/(x^n+1)$	64	7681	9	1712	2^{-128}

standardized KEMs ML-KEM and HQC, as well as the recent lightweight KEM Rudraksh, under the NIST security level 1 parameter sets.

The results show that, except for BAT, CTRU-Light achieves the smallest bandwidth. Furthermore, by leveraging NTT-friendly rings, our scheme achieves a highly compact polynomial arithmetic structure. Compared to module lattice-based schemes, it reduces public key storage by about 30% and computational overhead by roughly 50%. In contrast to traditional NTRU schemes defined over NTT-unfriendly rings, it enables efficient polynomial operations while maintaining performance comparable to recent cyclotomic-ring-based NTRU variants such as NEV. Although BAT achieves lower bandwidth at dimension $n = 512$, its key-generation procedure is several orders of magnitude slower than those of other NTRU-based KEMs. As a result, it fails to achieve competitive overall performance in protocols that require ephemeral key generation.

4 Lightweight Implementation on 8-bit AVR

4.1 NTT-based Polynomial Multiplication and Inversion

Benefiting from the introduction of NTT-friendly rings, we utilize incomplete NTT to compute the polynomial multiplication and inversion in $\mathcal{R}_q$.

4.2 Transformation

An important property of $q = 769$ used in CTRU-Light is that $q \equiv 1 \pmod{256}$, which guarantees the existence of a primitive 256-th root of unity in $\mathbb{Z}_q$. For $n = 512$, we have $\zeta^{n/4} \equiv -1 \pmod{q}$. Consequently, the radix-2 FFT factorization can be applied as

$$x^n + 1 = x^n - \zeta^{n/4} = (x^{n/2} - \zeta^{n/8}) \times (x^{n/2} + \zeta^{n/8}).$$

By iteratively applying seven levels of the radix-2 FFT factorization, the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ admits the following CRT isomorphism:

$$\mathbb{Z}_q[x]/(x^n+1) \cong \prod_{i=0}^{n/4-1} \mathbb{Z}_q[x]/(x^4 - \zeta^{2i+1}).$$

4.3 Base Case Multiplication

For the base case multiplication, resulting from the CRT decomposition, we need to compute 128 products of the form $c = a \cdot b \mod (x^4 - \omega), \omega = \zeta^{2i+1}$, where $a, b, c \in \mathbb{Z}_q[x]/(x^4 - \omega)$.

4.4 Base Case Inversion

The polynomial product $c = a \cdot b \bmod (x^4 - \omega)$ can be formulated as a matrix-vector multiplication over $\mathbb{Z}_q$, using the 4×4 rotation matrix $\mathbf{A}$ corresponding to a.

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & \zeta a_3 & \zeta a_2 & \zeta a_1 \\ a_1 & a_0 & \zeta a_3 & \zeta a_2 \\ a_2 & a_1 & a_0 & \zeta a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

To compute the inverse of a modulo $(x^4 - \omega)$, we first compute the inverse of the rotation matrix $\mathbf{A}$. This inverse is obtained by dividing the adjugate matrix by the determinant. If the determinant is zero, the polynomial is not invertible and the computation is aborted. Since the inverse matrix corresponds to the rotation matrix of the inverse polynomial, the coefficients of a^{-1} can be directly read from the first column. Specifically, we have $a^{-1} = |\det|^{-1}(a'_0 + a'_1 x + a'_2 x^2 + a'_3 x^3)$, where each a'_i denotes the algebraic cofactor corresponding to a_i in $\mathbf{A}$. Here, $|\det|$ denotes the determinant of the 4×4 rotation matrix $\mathbf{A}$. Consequently, computing $a^{-1} \bmod (x^4 - \omega)$ requires a single inversion of $|\det| \in \mathbb{Z}_q$, which is efficiently performed using Fermat's Little Theorem.

4.5 NTT-friendly Modular Multiplication

In lattice-based cryptography, integer multiplications are typically carried out modulo $\mathbb{Z}_q$. In this work, we present a technique leveraging the properties of NTT-friendly moduli to accelerate modular multiplication, which can be applied efficiently to CTRU-Light, Kyber, and Saber on 8-bit AVR devices.

4.5.1 Improving Montgomery Multiplication with NTT-friendly Moduli

Montgomery Multiplication. Assume $R = 2^m$, and let q be an odd integer such that $0 < q < \frac{R}{2}$. Algorithm 9 provides a mathematical description of the signed Montgomery multiplication as proposed in [Sei18].

Algorithm 9 Signed Montgomery multiplication [Sei18]

Input: $0 < q < \frac{R}{2}$ odd, $-\frac{R}{2} \le a < \frac{R}{2}$, $-q \le b \le q$, $q' = q^{-1} \mod R$
Output: $r' \equiv abR^{-1} \pmod q$, $-q < r' < q$
1: $t = a \cdot b = t_h \cdot R + t_l$ ▷ signed long multiplication
2: $m \leftarrow t_l q' \bmod^{\pm} R$ ▷ signed low multiplication, q' precomputed
3: $t_1 \leftarrow \lfloor \frac{mq}{R} \rfloor$ ▷ signed high multiplication
4: $r' \leftarrow t_h - t_1$

For two m-bit integers a and b, an instruction is referred to as a long multiplication if it computes the $2m$-bit product ab. A high multiplication is defined as an instruction that produces the upper m bits of ab, while a low multiplication produces the lower m bits. In Algorithm 9, the signed Montgomery multiplication is implemented by combining a long multiplication, a high multiplication, and a low multiplication.

Given the modulus $q = q_h \cdot 2^8 + q_l$ with $0 \le q_l < 2^8$, the red and black components in Algorithm 10 present the 8-bit AVR implementation of the signed Montgomery multiplication when $R = 2^{16}$ ($R = 2^{16}$ is assumed unless stated otherwise in following discussion). In this implementation, the high and low multiplications are replaced by a signed-unsigned long multiplication. This replacement primarily arises from the lack of advanced high/low multiplication instructions in the 8-bit AVR architecture, such as `smulh` in NEON or `mullo`/`mulhi` in AVX2. Consequently, implementing 16-bit high/low multiplications using

only the available 8×8 long multiplication instruction incurs cycles nearly identical to that of a full long multiplication, particularly in the case of high multiplication.

Algorithm 10 Signed Montgomery and NTT-friendly multiplication on 8-bit AVR

Input: $a = (\texttt{a1} \parallel \texttt{a0})$, $zR = (\texttt{z1} \parallel \texttt{z0})$, $\texttt{zero} = 0$,
`hi1,hi0,lo1,lo0,tmp3,tmp2,tmp1,tmp0,qhi,qlo`
Output: $a \cdot z = (\texttt{hi1} \parallel \texttt{hi0})$

```
1: ldi    qhi,   q_h                                  ▷ The black is common.
2: ldi    qlo,   q_l
3: muls16x16_32   a1,a0,z1,z0,zero,hi1,hi0,lo1,lo0
4: mulsu16x16_32   lo1,lo0,qhi,qlo,zero,tmp3,tmp2,a1,a0
5: mulsu16x16_lo16   lo1,lo0,qhi
6: ldi    qhi,   q'_h
7: ldi    qlo,   q'_l
8: mulsu16x16_32   a1,a0,qhi,qlo,zero,tmp3,tmp2,tmp1,tmp0
9: mulsu16x16_hi16   lo1,lo0,qhi,zero,tmp1,tmp0
10: sub   hi1,   tmp3
11: sbc   hi0,   tmp2
12: sub   hi1,   tmp1
13: sbc   hi0,   tmp0
```

Algorithm 11 `mulsu16x16_32`: 16×16 signed long multiplication on 8-bit AVR

Input: $a = (\texttt{ahi}\|\texttt{alo})$, $b = (\texttt{bhi}\|\texttt{blo})$, $\texttt{zero}=0$, `c3,c2,c1,c0`
Output: $a \cdot b = (\texttt{c3}\|\texttt{c2}\|\texttt{c1}\|\texttt{c0})$

```
1: muls ahi, bhi   4: movw  c0, r0     7: add c1, r0     10: mul bhi, alo
2: movw c2, r0     5: mulsu ahi, blo   8: adc c2, r1     11: add c1, r0
3: mul  alo, blo   6: sbc   c3, zero   9: adc c3, zero   12: adc c2, r1
                                                         13: adc c3, zero
```

Algorithm 12 `mulsu16x16_hi16`: 16×16 NTT-friendly high multiplication on 8-bit AVR

Input: $a = (\texttt{ahi}\|\texttt{alo})$, $b_h = \texttt{bhi}$, $\texttt{zero}=0$, `c1,c0`
Output: $(a \cdot b)_h = (\texttt{c1}\|\texttt{c0})$

```
1: muls  bhi, ahi   4: add r0, ahi    7: lsl ahi
2: movw  c0, r0     5: adc c0, r1     8: sbc c0, zero
3: mulsu bhi, alo   6: adc c1, zero   9: sbc c1, zero
```

Algorithm 13 `mulsu16x16_lo16`: 16×16 NTT-friendly low multiplication on 8-bit AVR

Input: $a = (\texttt{hi} \parallel \texttt{lo})$, $b_h = \texttt{bhi}$,
Output: $(a \cdot b)_l = (\texttt{hi} \parallel \texttt{lo})$

```
1: mulsu   bhi,   lo
2: add   hi,   r0
```

It is observed that the signed–unsigned 16-bit long multiplication in Algorithm 11 requires 17 cycles and is among the most time-consuming operations in the standard Montgomery multiplication, contributing 34 out of the total 58 cycles. Therefore, the primary contribution of this work is the introduction of a method that leverages the properties of NTT-friendly moduli to optimize the high and low multiplications in Montgomery multiplication. Specifically, our optimized high and low multiplication methods for 8-bit

AVR are detailed in Algorithms 12 and 13, respectively [2]. These algorithms are applicable to NTT-friendly moduli that support at least a primitive 256-th root of unity, thereby covering a broad range of practical parameter sets used in lattice-based cryptography.

Theorem 5. *For any NTT-friendly modulus q that admits at least a primitive 256-th root of unity, Algorithms 12 and 13 compute the signed high and low multiplications of a and q or q', respectively.*

Proof. The key insight in the proof of Theorem 5 is that NTT-friendly moduli, as well as their inverses in $\mathbb{Z}_{2^m}$, can be expressed in a specific high- and low-bit representation.

The proof begins with the computational steps of standard long multiplication. Consider two 16-bit integers a and b, which can be expressed in two-word form as $a = a_h \cdot \sqrt{R} + a_l$ and $b = b_h \cdot \sqrt{R} + b_l$, where $-\frac{\sqrt{R}}{2} < a_h, b_h < \frac{\sqrt{R}}{2}$ and $0 \le a_l, b_l < \sqrt{R}$. A 16-bit long multiplication takes a and b as input and computes their 32-bit product $c = a \cdot b$.

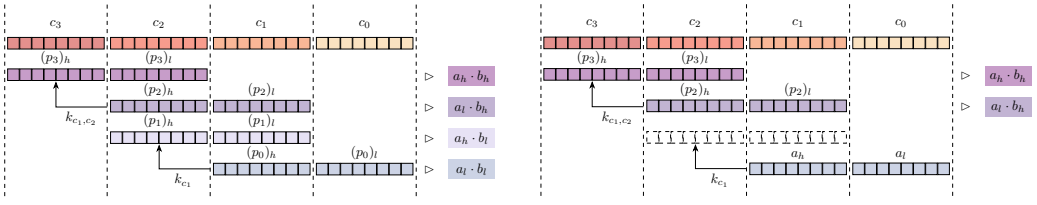
Figure 1a illustrates the 16-bit long multiplication. On the 8-bit AVR platform, a 16-bit long multiplication is carried out using four 8×8 long multiplications, since the architecture provides only 8-bit operations. These four 8×8 multiplications are performed as follows, each producing a 16-bit intermediate product represented in two words:

$$\begin{aligned} p_0 &= (p_0)_h \cdot \sqrt{R} + (p_0)_l = a_l \cdot b_l \\ p_1 &= (p_1)_h \cdot \sqrt{R} + (p_1)_l = a_l \cdot b_h \\ p_2 &= (p_2)_h \cdot \sqrt{R} + (p_2)_l = a_h \cdot b_l \\ p_3 &= (p_3)_h \cdot \sqrt{R} + (p_3)_l = a_h \cdot b_h. \end{aligned}$$

Specifically, the 32-bit product $c = a \cdot b$ can be expressed in four words as $c = c_3 \cdot R\sqrt{R} + c_2 \cdot R + c_1 \cdot \sqrt{R} + c_0$, where $-\frac{\sqrt{R}}{2} < c_3 < \frac{\sqrt{R}}{2}$ and $0 \le c_2, c_1, c_0 < \sqrt{R}$. Following an approach analogous to polynomial multiplication, each word of c is given by

$$\begin{aligned} c_0 &= (p_0)_l \\ c_1 &= (p_0)_h + (p_1)_l + (p_2)_l \\ c_2 &= (p_1)_h + (p_2)_h + (p_3)_l + k_{c_1} \\ c_3 &= (p_3)_h + k_{c_1,c_2}, \end{aligned}$$

where k_i denotes the carry correction from the computation of lower words. It can be observed that the upper two words of c (i.e., the result of 16-bit high multiplication) depend on all four 8×8 multiplications, while the lower two words (i.e., the result of 16-bit low multiplication) depend on three of them. This explains why implementing high and low multiplications incurs approximately the same computational cost as long multiplication.



(a) 16-bit long multiplication for arbitrary inputs

(b) 16-bit long multiplication for NTT-friendly inputs

Figure 1: 16-bit Long Multiplication: General vs. NTT-friendly Inputs

However, if one of the 16-bit factors is constrained to an NTT-friendly modulus q or q' (i.e., the inverse of q in $\mathbb{Z}_R$), the computation becomes simpler.

[2] In the AVR architecture, the result of a multiplication instruction is stored in the register pair `r1:r0`, and we maintain this representation in Algorithms 12 and 13.

Specifically, assuming q is an NTT-friendly modulus that admits at least a primitive 256-th root of unity, it must hold that $256 \mid (q-1)$. Then q can be written as

$$q = k \cdot 256 + 1 = q_h \cdot \sqrt{R} + q_l,$$

where $q_h = k$, $-\frac{\sqrt{R}}{2} < k < \frac{\sqrt{R}}{2}$, and $q_l = 1$.

On the other hand, since $q' = q^{-1} \bmod R$, we have

$$q' = 1 - k \cdot 256 = q'_h \cdot \sqrt{R} + q'_l,$$

where $q'_h = -k$ and $q'_l = 1$. Therefore, both q and q' can be regarded as special 16-bit integers b of the form

$$b = b_h \cdot \sqrt{R} + b_l = b_h \cdot \sqrt{R} + 1.$$

In this situation, the four 8×8 multiplications in the 16-bit long multiplication $c = a \cdot b$ reduce to:

$$\begin{aligned} p_0 &= a_l, \\ p_1 &= (p_1)_h \cdot \sqrt{R} + (p_1)_l = a_l \cdot b_h, \\ p_2 &= a_h, \\ p_3 &= (p_3)_h \cdot \sqrt{R} + (p_3)_l = a_h \cdot b_h. \end{aligned}$$

Then each word of c is given by

$$\begin{aligned} c_0 &= a_l, \\ c_1 &= (p_1)_l + a_h, \\ c_2 &= (p_1)_h + (p_3)_l + k_{c_1}, \\ c_3 &= (p_3)_h + k_{c_1,c_2}, \end{aligned}$$

where k_i denotes the carry correction. In this case, the upper words of c depend on two 8×8 multiplications, while the lower words depend on only one.

Figure 1b illustrates this computation, which corresponds to the steps in Algorithms 12 and 13, computing the signed high and low multiplications of an integer a and q (or q') in this special form, respectively.

□

Therefore, Algorithms 12 and 13 can be used to perform the high and low multiplications involving a factor of q or q' in Algorithm 9, thus replacing the signed–unsigned long multiplication operations in Algorithm 10. The black and blue components in Algorithm 10 describe the NTT-friendly Montgomery multiplication, an improved Montgomery multiplication that incorporates optimized high and low multiplications. Since Algorithms 12 and 13 require only 11 and 3 cycles, respectively, the entire NTT-friendly Montgomery multiplication can be completed in just 36 cycles, yielding a 38% speedup over the original Montgomery multiplication.

4.5.2 Approximate Equivalence between K-RED2X and Montgomery Multiplication under NTT-Friendly Moduli

While the NTT-friendly Montgomery algorithm can be applied to small moduli, the K-RED2X method proposed in [LN16] offers better efficiency in this setting. We further show that the two methods are approximately equivalent under NTT-friendly moduli. Specifically, the outputs of Montgomery reduction and K-RED2X are congruent modulo q, and the absolute difference between them is bounded by $(k-1)q$, where $q = k \cdot 2^8 + 1$.

Algorithm 14 K-RED2X multiplication [LN16]

Input: $a, b \in \mathbb{Z}, q = k \cdot 2^m + 1$
Output: $r' \equiv k^2 \cdot ab \pmod{q}$
1: $c = a \cdot b$
2: $c_0 = c \mod 2^m$
3: $c_1 = c/2^m \mod 2^m$
4: $c_2 = c/2^{2m} \mod 2^m$
5: $r' = k^2 c_0 - k c_1 + c_2$

Let a and b be integers, and assume $q = k \cdot 2^m + 1$ for a small integer $k \geq 3$. Algorithm 14 gives a mathematical specification of the K-RED2X multiplication.

Theorem 6. *Under NTT-friendly moduli admitting a 256-th root of unity, the K-RED2X multiplication can be regarded as a variant of Montgomery multiplication. Suppose* $-\frac{R}{2} \leq a < \frac{R}{2}$ *and* $-q \leq b \leq q$*, the results of K-RED2X multiplication and Montgomery multiplication satisfy the following relation:*

$$\begin{aligned} r_{\mathrm{KRED}} &= \mathbf{K-RED2X}_q(ab) = abk^2 \bmod q, \\ r_{\mathrm{MONT}} &= \mathbf{Mont}_q(ab) = abR^{-1} \bmod q, \end{aligned}$$

$$\begin{aligned} r_{\mathrm{KRED}} &\equiv r_{\mathrm{MONT}} \bmod q, \\ |r_{\mathrm{KRED}} - r_{\mathrm{MONT}}| &\leq (k-1)q. \end{aligned}$$

Proof. The proof primarily involves carefully unraveling the definition in conjunction with the properties of NTT-friendly moduli.

Suppose $c = c_2 \cdot 2^{2m} + c_1 \cdot 2^m + c_0 = a \cdot b, m = 8$ and $R = 2^{2m}$, where

$$c_0 = c \mod 2^m, c_1 = c/2^m \mod 2^m, c_2 = \lfloor c/2^{2m} \rfloor.$$

Then we have

$$\begin{aligned} r_{\mathrm{MONT}} &= \mathbf{Mont}_q(c) \\ &= \lfloor \frac{c}{R} \rfloor + \lfloor \frac{q \cdot (-q^{-1} \cdot c \bmod R)}{R} \rfloor \\ &= c_2 + \lfloor \frac{q \cdot (-q^{-1} \cdot (c_1 \cdot 2^m + c_0) \bmod R)}{R} \rfloor. \end{aligned}$$

Assuming that q is an NTT-friendly modulus admitting at least a primitive 256-th root of unity, it follows that $256|(q-1)$. Consequently, q can be expressed as $q = k \cdot 2^m + 1$, which also implies $-q^{-1} = k \cdot 2^m - 1$. Therefore, the above formula can be written as

$$\begin{aligned} c_2 + \lfloor \frac{q \cdot (-q^{-1} \cdot (c_1 \cdot 2^m + c_0) \bmod R)}{R} \rfloor &= c_2 + \lfloor \frac{q \cdot ((k \cdot 2^m - 1) \cdot (c_1 \cdot 2^m + c_0) \bmod R)}{R} \rfloor \\ &= c_2 + \lfloor \frac{q \cdot (kc_1 \cdot 2^{2m} + 2^m(kc_0 - c_1) - c_0 \bmod R)}{R} \rfloor \\ &= c_2 + \lfloor \frac{q \cdot (2^m(kc_0 - c_1) - c_0 \bmod R)}{R} \rfloor. \end{aligned}$$

For the integer $2^m(kc_0 - c_1) - c_0 \in \mathbb{Z}$, there exist $r \in \mathbb{Z}_{2^{2m}}$ and $e \in \mathbb{Z}$ such that

$$\begin{aligned} r &\equiv 2^m(kc_0 - c_1) - c_0 \bmod R, \\ r + eR &= 2^m(kc_0 - c_1) - c_0. \end{aligned}$$

Hence, we obtain

$$\begin{aligned} c_2 + \lfloor \frac{q \cdot (2^m(kc_0 - c_1) - c_0 \bmod R)}{R} \rfloor &= c_2 + \lfloor \frac{q \cdot (2^m(kc_0 - c_1) - c_0) - q \cdot eR}{R} \rfloor \\ &= c_2 + \lfloor \frac{(k \cdot 2^m + 1) \cdot (2^m(kc_0 - c_1) - c_0)}{R} \rfloor - eq \\ &= c_2 + \lfloor \frac{(k^2c_0 - kc_1) \cdot 2^{2m} + (kc_0 - c_1) \cdot 2^m - c_0(k \cdot 2^m + 1)}{R} \rfloor - eq \\ &= c_2 + \lfloor \frac{(k^2c_0 - kc_1) \cdot 2^{2m} - c_1 \cdot 2^m - c_0}{R} \rfloor - eq \\ &= k^2c_0 - kc_1 + c_2 - eq \\ &= r_{\mathrm{KRED}} - eq \end{aligned}$$

Since $|e| = |\lfloor \frac{2^m(kc_0 - c_1) - c_0}{R} \rfloor| \leq k - 1$, Theorem 6 is concluded. □

Algorithm 15 K-RED2X multiplication on 8-bit AVR

Input: $a = (\texttt{al}\|\texttt{ah})$, $bk^{-2} = (\texttt{bl}\|\texttt{bh})$, `z`=0, `rl,rh,t0,t1,tmp0,tmp1`
Output: $a \cdot b = (\texttt{rl}\|\texttt{rh})$

```
1: muls16x16_32 al,ah,bl,bh,z,rl,rh,t0,t1
2: ldi tmp0, 0x09
3: ldi tmp1, 0x03
4: mul rl, tmp0
5: mov  al, rh
6: movw rl, r0
7: mul  al, tmp1
8: sub  rl, r0
9: sbc rh, r1
10: add rl, t0
11: adc rh, t1
```

Given that the results of Montgomery multiplication are bounded in magnitude by q, the results of K-RED2X multiplication are bounded in magnitude by kq, according to Theorem 6. In the NTT, each butterfly layer can increase the bound on coefficient magnitudes by at most kq, and the coefficients must remain strictly less than $\frac{R}{2}$ in magnitude throughout the computation. Therefore, a 16-bit modular reduction algorithm (e.g., Barrett reduction) is required every d-th layer in order to ensure that $(dk+1)q < \frac{R}{2}$. Consequently, using K-RED2X multiplication in NTT requires more frequent modular reductions for larger moduli, thereby reducing efficiency. Hence, our design adopts the K-RED2X multiplication method exclusively in small-modulus scenarios. For the small modulus $q = 769 = 3 \times 2^8 + 1$ in CTRU-Light, single word reduction is not required throughout the 7 butterfly layers, since $(dk+1)q < \frac{R}{2}$ holds for $k = 3$ and $d = 7$. However, for the larger modulus $q = 3329 = 13 \times 2^8 + 1$ in CTRU-Light, single word reduction is required in each butterfly layer, because $(k+1)q > \frac{R}{2}$. This observation highlights that the small-modulus nature of our algorithm design provides advantages in terms of engineering practicality and implementation efficiency.

4.5.3 Improving Barrett Multiplication with NTT-friendly Moduli

The properties of NTT-friendly moduli can also be utilized to optimize Barrett multiplication. Let $a \in [-\frac{R}{2}, \frac{R}{2})$ and $b \in [-\frac{q}{2}, \frac{q}{2})$. The Barrett multiplication proposed by Hwang *et al.* [HKS25] computes $ab \bmod q$ as

$$ab - \left\llbracket \frac{a \left\llbracket \frac{bR}{q} \right\rrbracket_0}{R} \right\rrbracket_1 \cdot q,$$

where $\left\llbracket \frac{bR}{q} \right\rrbracket_0$ is precomputed. Since only the lower half of $\left\llbracket \frac{a \left\llbracket \frac{bR}{q} \right\rrbracket_0}{R} \right\rrbracket_1 \cdot q$ is required to obtain the final result, our optimized low-half multiplication (Algorithm 13) can be

efficiently applied when using an NTT-friendly modulus q. With different choices of the approximate computations, Barrett multiplication can be implemented with varying upper bounds and computational efficiency. Among these variants, the approximate version (where $[\![\,]\!]_0 = \lfloor\,\rceil$ and $[\![\,]\!]_1$ is defined as described in [HKS25]) offers the simplest computation and shares the same lazy-reduction sequence as K-RED2X under the modulus 769. Therefore, we take it as an example.

Algorithm 16 illustrates the process of Barrett modular multiplication modulo 769, optimized using the NTT-friendly method. In the algorithm, `muls16x16_lo16*` denotes the unoptimized low multiplication. Steps corresponding to the original Barrett multiplication are shown in red, while the optimized computations are highlighted in blue. With the NTT-based optimization, the Barrett multiplication reduces the usage of 3 fixed registers and saves 6 clock cycles. In fact, our optimization approach can be applied to all variants of Barrett multiplication, providing the same level of improvement.

Algorithm 16 Barrett Multiplication Optimized for NTT-Friendly Moduli on 8-bit AVR

Input: $a = (\texttt{al}\|\texttt{ah})$, $b = (\texttt{bl}\|\texttt{bh})$, $\left[\!\left[\frac{bR}{q}\right]\!\right]_0 = (\texttt{bpl}\|\texttt{bph})$, $\texttt{z} = 0$, `rl`, `rh`, `brt0`, `brt1`, `brt2`, `brt3`, `c_tmp`, `cst_q`

Output: $a \cdot b = (\texttt{rl}\|\texttt{rh})$

```
1: muls16x16_lo16* al,ah,bl,bh,z,rl,rh                    ▷ The black is common
2: muls  ah,   bph     5: add   brt0, r1       8: add brt0,  r1
3: movw  brt0, r0      6: adc   brt1, z        9: adc brt1,  z
4: mulsu ah,   bpl     7: mulsu bph,  al      10: ldi cst_q, 0x0D
11: ldi c_tmp,  0x01   12: muls16x16_lo16* brt0,brt1,c_tmp,cst_q,z,brt2,brt3
13: sub   rl,   brt2   14: sbc   rh,   brt3       ▷ The red is based on [HKS25]
11: muls16x16_lo16 brt0,brt1,cst_q
12: sub   rl,   brt0   13:  sbc   rh,   brt1      ▷ The blue is our NTT-Friendly
```

4.5.4 Comparison of Modular Multiplications

Table 3 summarizes the cycle counts, register usage, and memory consumption of various modular reduction methods on an 8-bit AVR device. For a fair comparison, all implementations were written in handwritten AVR assembly with redundant operations removed. Both our NTT-friendly Montgomery and K-RED–based multiplication methods outperform all previous approaches without incurring additional flash memory overhead. While our optimized Barrett multiplication achieves the fewest cycles, it requires one additional precomputed table.

Table 3: Comparisons of 16-bit modular multiplication on 8-bit AVR.

Name	Type	Register usage	Cycles	Precomputation
Solinas multiplication [YMÖS21]	unsigned	14	94	1 zetas
Plantard multiplication [HZZ+22]	signed	18	74	1 zetas
LookUp-Table multiplication [SLP+17]	unsigned	11	57	1 zetas + 2 LUT
Montgomery multiplication [Sei18]	signed	15	50	1 zetas
Barrett multiplication [HKS25]	signed	15	38	2 zetas
Optimized Barrett multiplication (This work)	signed	12	26	2 zetas
Signed LUT multiplication† [KS25]	signed	9	45	1 zetas + 2 LUT
Signed LUT multiplication‡ [KS25]	signed	9	41	1 zetas + 2 LUT
NTT-Friendly multiplication (This work)	signed	12	36	1 zetas
K-RED2X multiplication (This work)	signed	11	30	1 zetas

†: placing LUTs in flash memory, ‡: placing LUTs in stack

Comparison with arithmetic methods. Our method optimizes Montgomery modular multiplication by leveraging the properties of NTT-friendly moduli, yielding clear advantages in both cycle counts and memory usage compared to the original Montgomery multiplication [Sei18] and its variant, the Plantard method [HZZ+22] (In fact, the Plantard multiplication is relevant in environments with 32-bit registers and powerful multiplication instructions such as `smulwb` and `smlabb`). In particular, Barrett multiplication requires a precomputed table of twiddle factors twice as large as those of other methods. This not only increases memory pressure on resource-constrained AVR platforms but also doubles the number of `ldd` instructions needed during NTT execution. Moreover, due to limited SRAM, larger twiddle tables are typically stored in flash memory, where data loading via `lpm` takes 3 cycles instead of 2 for SRAM loads, further exacerbating the runtime overhead of Barrett multiplication. Even though our optimized Barrett multiplication achieves the shortest execution time, it inevitably inherits the drawbacks of Barrett reduction.

Comparison with Look-Up Table methods. Compared with lookup table–based methods (i.e., the LookUp-Table multiplication [SLP+17] and the unsigned LUT multiplication [KS25]), our approaches achieve markedly lower cycle counts. The K-RED2X method requires the same number of registers as the LUT method used in Kyber [KS25], with the exception of the two constant registers that are reused in K-RED2X. In contrast, the NTT-friendly Montgomery and Barrett multiplication methods each consume three additional registers. However, register pressure is not a bottleneck in NTT and iNTT, so the extra registers yield no tangible performance benefits, as analyzed later. In addition, LUT-based modular multiplication requires maintaining 2 look-up tables whose sizes grow with the modulus, further exacerbating the memory constraints of AVR devices.

Comparison with Solinas multiplication. In [YMÖS21], Solinas multiplication exploits the linearity of the modulus to reduce the output range. For example, when $q = 3329$, since $3329 = 2^{12} - 2^9 - 2^8 + 1$, Solinas multiplication reduces the output size by replacing 2^{12} with $2^9 + 2^8 - 1$. However, this method remains inefficient unless q is a complete Mersenne prime, as general moduli such as 3329 or 769 require significant additional computations. Therefore, our NTT-friendly multiplication and K-RED2X method are more suitable in general settings, offering clear advantages in terms of both cycle counts and register usage.

Advantage of reduced register usage. In CTRU-Light, the base-case multiplication is implemented as four-dimensional polynomial products, requiring simultaneous arithmetic on eight 16-bit integers. Combined with the storage of intermediate results, this imposes significant pressure on register scheduling. Under classical Montgomery multiplication, some registers must be temporarily saved and restored via `push` and `pop` to accommodate multiple concurrent computations. In contrast, when modular multiplication requires fewer dedicated registers (as in the cases of NTT-friendly and K-RED2X multiplication), additional registers become available for scheduling, thereby reducing register pressure. The elimination of `push`/`pop` instructions further enhances computational efficiency.

In NTT and iNTT, the independence of butterfly operations across layers and branches enables efficient computation with limited register reuse, making register usage relatively unconstrained. While this avoids frequent `push`/`pop` operations, most register contents are overwritten during computation. Consequently, immediate constants that must be repeatedly used (e.g., q and q' in Montgomery multiplication) need to be reloaded via `ldi` instructions. By reducing the fixed register requirements of modular multiplication, the released registers can instead be reserved for permanent storage of such constants, thereby eliminating repeated `ldi` loads. In our implementation, this strategy is realized by globally loading constants once, instead of reloading them within each NTT-friendly or K-RED2X multiplication method, thereby further enhancing performance.

4.5.5 Selection of Optimization Strategies

Based on the NTT-friendly multiplication, we optimize not only our CTRU-Light but also the existing AVR implementations of Kyber and Saber. Since the optimized Barrett multiplication provides only marginal benefits while doubling the number of precomputed constants, our implementation focuses on the NTT-friendly Montgomery and K-RED2X approaches.

- For the modulus 3329 used in Kyber and the moduli involved in the multi-moduli polynomial multiplication of Saber, we employ the general NTT-friendly Montgomery modular multiplication. The proposed method is broadly applicable to moduli admitting a 256-th root of unity.
- For the modulus 769 used in CTRU-Light, we adopt the K-RED2X algorithm. The K-RED2X multiplication provides performance advantages over NTT-friendly Montgomery multiplication for small moduli, although such benefits diminish as the modulus size increases.

4.6 ASCON-based Symmetric Primitives

Similar to other lattice-based PQC KEMs, CTRU-Light employs extendable-output functions (XOFs) for random sampling and hash functions in the FO transformation, both relying on symmetric primitives. As symmetric components such as AES, GCM, SHA, and SHAKE have been identified as major performance bottlenecks on embedded platforms, we adopt the lightweight cryptography standard ASCON in our AVR implementation to improve efficiency under resource constraints.

The ASCON standard specifies three core algorithms: the ASCON-AEAD authenticated encryption algorithm, the ASCON-Hash algorithm, and the ASCON-XOF algorithm. Notably, ASCON-XOF includes a variant, ASCON-CXOF, which allows users to specify a customization string and select the output length of the message hash. In the instantiation of the symmetric primitives within CTRU-Light, we use ASCON-XOF for pseudorandom number generation in sampling and ASCON-Hash to instantiate the hash functions H and G, all providing 128-bit security. Our implementation employs the speed-optimized 8-bit AVR version of ASCON[3].

4.7 Constant-time Implementation

Our C implementation contains no secret-dependent conditional branches, memory accesses and divisions. Our implementation of the scalable E_8 lattice code follows the same constant-time design principles as in [LFZZ24b]. Following the KyberSlash case [BBB+24a], we also ensure that message encoding, packing, and related routines are implemented without using division, so that the execution time does not vary with the input values. However, the optimized Kyber implementation used in [KS25] does not include the patches introduced by KyberSlash. We therefore additionally apply these patches and release an updated AVR version of Kyber. The 8-bit AVR instruction set contains no variable-time instructions other than conditional branches. Multiplication is supported by an 8-bit multiply-long instruction that executes in constant time (`mul muls mulsu`, 2 cycles). Since AVR devices operate in cacheless environments, they are unaffected by cache-timing attacks; hence, LUT-based reduction can be safely employed.

[3]Reference, highly optimized, masked C and ASM implementations of Ascon (NIST SP 800-232), available at https://github.com/ascon/ascon-c.

5 Hybrid Lightweight MQTT Protocol for 8-bit AVR

5.1 Hybrid KEM-MQTT

Figure 2 illustrates the design of the Hybrid KEM-MQTT protocol. Similar to the PDK-based KEM-MQTT model [KS25], the AVR devices in Hybrid KEM-MQTT function solely as publishers (i.e., clients, typically sensor nodes in IoT environments) and communicate exclusively with the broker (a gateway that manages subscriptions and message distribution). Hybrid KEM-MQTT preserves the following key properties:

No certificates: For AVR sensor nodes, the broker's public key is preloaded into flash memory before deployment. Likewise, the broker is treated as a semitrusted entity, and is preconfigured with the public key of each sensor node. Therefore, authentication in the protocol does not rely on certificates but instead on the reliable identification of each sensor node's public key.

Mutual authentication: To prevent sensor node impersonation by malicious attackers, the Hybrid KEM-MQTT scheme authenticates both the publisher and the broker. Similar to KEMTLS, it achieves implicit mutual authentication through KEM/ECDH instead of signatures, and establishes explicit authentication upon receiving the key confirmation message.

Confidentiality and integrity: To ensure confidentiality and integrity, published data and handshake messages are encrypted using Authenticated Encryption (AE). Furthermore, the protocol allows the transmission of associated data (e.g., the publisher ID and the topic), which are protected using an Authenticated Encryption with Associated Data (AEAD) algorithm. Although the associated data itself is not encrypted, its integrity is guaranteed.

The light blue sections in Figure 2 illustrate the combination of classical key exchange schemes (e.g., ECDH) and post-quantum KEMs to enable hybrid operation. Before the handshake, both parties possess static post-quantum and classical key pairs, and each is assumed to know the other's public keys. When the handshake starts, the protocol proceeds as follows:

- *Ephemeral key exchange:* After establishing the connection, the publisher begins to send the ephemeral public keys, pk_u and g^{x_u}, in the `ClientHello` message. Upon receipt, the server responds with the `ServerHello`. It includes an encapsulation result ct_u against pk_u, as well as the broker's ephemeral public keys pk_v and g^{x_v}. Then the publisher produces and transmits an encapsulation result ct_v against pk_v. At this stage, both parties have established three independent ephemeral shared secrets: ss_u, ss_v, and ss_{ecdh_t}.
- *Implicitly authenticated key exchange:* At the start of the handshake, the publisher derives shared secrets with the broker's static public keys. It sends the resulting ciphertext ct_b and its ECDH public key g^{x_p} in the `ClientHello` message. This yields implicitly authenticated shared secrets ss_b and ss_{ecdh_s}. Conversely, the publisher performs a similar operation to authenticate the broker and derive an additional shared secret ss_p associated with its static key pk_p.
- *Confirmation / explicit authentication:* Both parties exchange confirmation messages, and the explicit authentication is completed once these messages are received.

Throughout the above process, the protocol follows the same key schedule as KEMTLS [SSW20]. In hybrid mode, the shared secrets established in each step are concatenated prior to key derivation (a design choice consistent with NIST standards for key derivation [BCD+18]), ensuring that every shared secret contributes to the generation of the final traffic keys. In this process, both the publisher and broker must support all algorithms involved in the selected hybrid mode. In resource-constrained AVR environments, this requirement can be addressed by instantiating only one classical algorithm and one

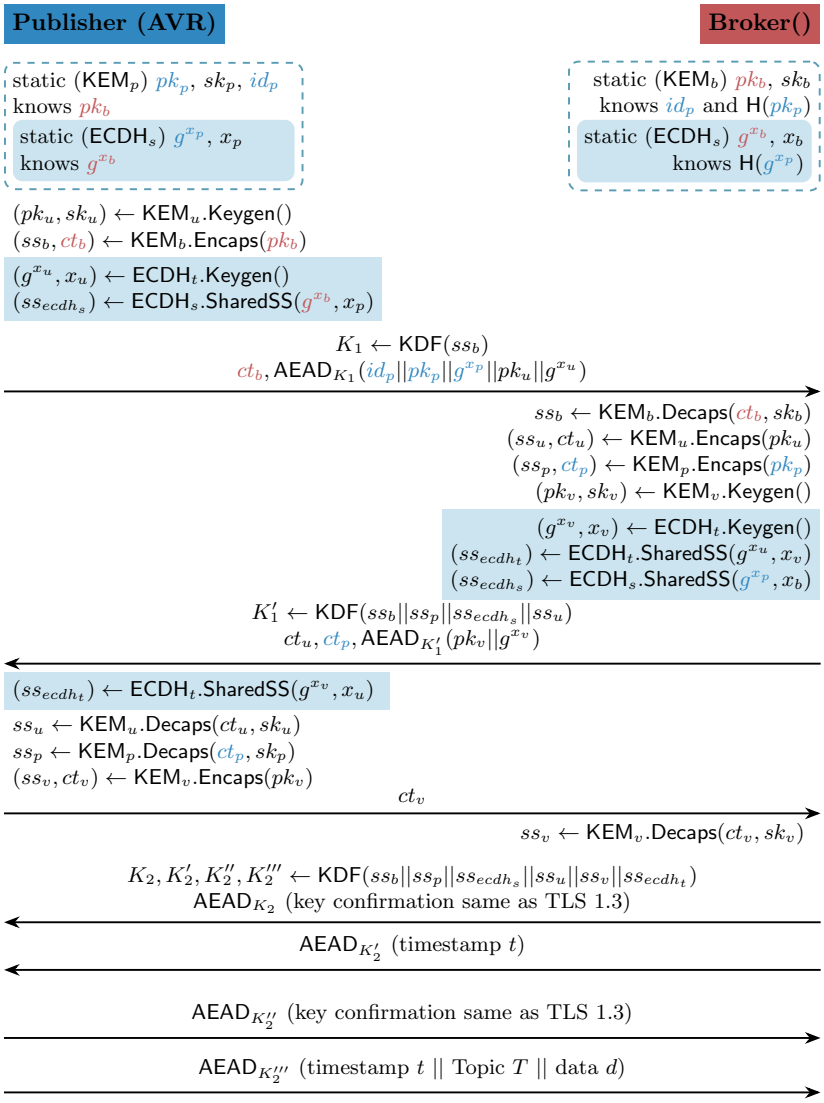


Figure 2: Sketch of Hybrid KEM-MQTT Handshake.

post-quantum algorithm at a time, since the design allows ephemeral key exchange and authentication to use the same algorithm.

5.2 Security Analysis

As a variant of KEMTLS-PDK, KEM-MQTT inherits essential security properties, including forward secrecy, explicit authentication, and key-use guarantees. These properties can be formally verified using the multistage security proof methodology of KEMTLS-PDK [SSW21]. Hybrid KEM-MQTT can be regarded as a hybrid mode of KEM-MQTT. We follow the KEMTLS design for both KEM and ECDH operations. This preserves the original protocol's message flow and keeps the existing security analysis intact. From an intuitive perspective, the key security properties of the Hybrid KEM-MQTT protocol can be described as follows:

- *Forward secrecy*: Similar to TLS 1.3 [DFGS21] and KEMTLS [SSW20], Hybrid KEM-

MQTT enforces the use of ephemeral shared secrets to ensure forward secrecy. In our protocol, these secrets comprise one classical ECDH key and two post-quantum keys, encapsulated separately by the publisher and the broker. Compared to KEM-MQTT, our scheme adds an additional ephemeral post-quantum key from the broker to mitigate potential attacks by malicious publishers.

- *Key indistinguishability*: The secrecy of each stage key is guaranteed by its indistinguishability from random, as defined in [BR93]. This property follows from the underlying security of the post-quantum KEMs and ECDH.
- *Authentication*: Our scheme adopts a mechanism analogous to the TLS 1.3 PDK scenario [ST16], leveraging cached information to bypass server certificate verification. Although this approach has not been widely used in classical settings, it is advantageous in our work due to the large post-quantum certificate sizes and the resource constraints of AVR-based devices. Requiring the broker to store all certificates would impose a significant burden. Therefore, we propose that the broker store only 32-byte hashes of public keys. Sensor nodes can then be uniquely identified through these public key hashes, thereby enabling explicit authentication.

6 Experimental Results

6.1 Benchmarking Setup

We benchmark our implementations using `Microchip Studio 7.0`. For CTRU-Light, we select the ATmega4808 as the target platform, which provides 6 KB of SRAM and 48 KB of flash memory. The ATmega4808 is widely adopted in wireless sensor networks and allows precise measurement of cycle counts and memory usage [KS25]. For comparison, we evaluate Kyber and Saber on the ATmega1284P. While the ATmega1284P integrates 16 KB of SRAM and 128 KB of flash memory, its newer counterpart, the ATmega4808, provides enhanced peripheral support and a slightly improved memory architecture. This configuration enables more computationally demanding lattice-based PQC schemes such as Kyber and Saber. All implementations are compiled with `avr-gcc 14.1` under Microchip Studio using the `-O3` optimization flag and a 20 MHz clock. For a fair comparison, we also port the LightSaber implementation from [HKS25] to `Microchip Studio` to ensure a consistent benchmarking environment.

6.2 Performance Results of Polynomial Arithmetic

We evaluate the performance of polynomial arithmetic across three schemes: CTRU-Light, Kyber, and Saber. Among these, Saber employs multi-moduli polynomial multiplication to accelerate NTT-unfriendly polynomial operations, which are operated over two polynomial rings. For CTRU-Light, because there is no prior AVR implementation, we also provide a comparison between the C reference implementation and our optimized AVR implementation. Furthermore, we integrate various optimized polynomial arithmetic modules into each scheme and measure the resulting acceleration, demonstrating the impact of these optimizations on overall scheme performance. Please see Tables 4 and 5 for a summary.

Comparison of Modular Multiplication without Extra Flash Usage. We evaluate the efficiency of our proposed NTT-friendly Montgomery multiplication and K-RED2X (for modulus 769) against the classical Montgomery method, focusing on designs that require no additional flash memory. The results reveal clear advantages of our approaches across different ring dimensions and modulus sizes. For the ring $\mathbb{Z}_{3329}[x]/(x^{256}+1)$ used in Kyber, our NTT-friendly multiplication achieves speedups of 13.4%, 41.3%, and 59.6% in the NTT, iNTT, and base-case multiplication, respectively. For the lightweight modulus 769 adopted

Table 4: Performance of Polynomial Multiplication under Different Modular Multiplication in Kyber, Saber, and CTRU-Light (kcycles).

Scheme	Ring	Implementation	NTT	iNTT	Base Mult	Base Inv
CTRU-Light	$\mathbb{Z}_{769}[x]/(x^{512}+1)$	Ref.	330.4	426.0	263.6	1593.6
		Montgomery†	131.1	161.5	117.3	621.4
		NTT-friendly†	115.0	140.8	101.5	518.9
		NTT-friendly‡	114.7	140.5	101.6	518.9
		K-RED2X†	108.1	127.2	99.7	439.4
Kyber	$\mathbb{Z}_{3329}[x]/(x^{256}+1)$	Montgomery‡ [KS25]	67.1	119.3	95.6	-
		Signed LUT† [KS25]	62.2	72.4	43.3	-
		Signed LUT‡ [KS25]	58.4	67.8	41.2	-
		NTT-friendly‡ (This work)	58.1	70.0	38.6	-
Saber	$\mathbb{Z}_{3329}[x]/(x^{256}+1)$	Barrett‡ [HKS25]	60.0	163.5	353.7	-
		NTT-friendly† (This work)	58.3	80.0	43.2	-
		NTT-friendly‡ (This work)	58.1	70.0	38.7	-
	$\mathbb{Z}_{7681}[x]/(x^{256}+1)$	Barrett† [HKS25]	70.3	188.7	197.5	-
		NTT-friendly† (This work)	67.7	89.6	21.9	-
		NTT-friendly‡ (This work)	67.2	91.5	24.3	-

†: placing LUTs/zetas in flash memory, ‡: placing LUTs/zetas in stack

by CTRU-Light, our NTT-friendly design still delivers consistent gains of 12.3%, 12.8%, 13.4%, and 16.5% in NTT, iNTT, base-case multiplication, and inversion, respectively. Meanwhile, the K-RED2X method achieves even higher improvements—17.5%, 21.2%, 15.1%, and 29.3%—surpassing the NTT-friendly approach by up to 15.3% under small-modulus configurations. Overall, these results confirm that carefully structured reduction strategies can substantially enhance modular multiplication efficiency on AVR devices, even without relying on precomputed tables or flash-expensive techniques.

Comparison with the state-of-the-art Modular Multiplication with Extra Flash Usage. State-of-the-art modular multiplication methods, such as Barrett multiplication (used in Saber) and signed LUT multiplication (used in Kyber), achieve high speed at the cost of additional flash consumption. Specifically, Barrett multiplication requires precomputed constants for reduction in the NTT, while signed LUT multiplication relies on two 256-byte lookup tables. In contrast, our NTT-friendly multiplication introduces *no additional flash overhead* while maintaining competitive, and often superior, performance. Compared to signed LUT multiplication, our approach achieves speedups of 6.6%, 3.3%, and 10.7% for NTT, iNTT, and base-case multiplication, respectively. For the multi-moduli polynomial multiplications used in Saber, the gains become even more pronounced: 3.3%, 57.2%, and 89.1% in $\mathbb{Z}_{3329}[x]/(x^{256}+1)$, and 4.4%, 51.5%, and 87.7% in $\mathbb{Z}_{7681}[x]/(x^{256}+1)$. These results highlight that, even without leveraging precomputed tables, our NTT-friendly method can substantially improve execution efficiency while preserving a compact memory footprint.

CTRU-Light AVR Implementation Modules. In CTRU-Light, we accelerate the main computational bottlenecks. These include NTT, iNTT, base-case multiplication, and base-case inversion. All optimizations target the strict hardware limits of 8-bit AVR microcontrollers. Our optimized assembly routines deliver substantial performance improvements, achieving speedups of 67.2% for the NTT, 70.1% for the iNTT, 62.2% for the base-case multiplication, and 72.4% for the base-case inversion. These results demonstrate that even within the limited register set and memory capacity of AVR devices, significant acceleration of operations is attainable through fine-grained instruction-level optimization.

Table 5: Performance of KEM Schemes under Different Modular Multiplication.

Scheme	Implementation	KeyGen (kcycles)	Encaps (kcycles)	Decaps (kcycles)	Stack (bytes)	Code Size (bytes)
CTRU-Light	Montgomery †	1835.7	1332.6	2936.4	2878	30584
	NTT-friendly†	1664.3	1263.8	2798.7	2878	28918
	NTT-friendly‡	1663.8	1263.1	2797.4	3134	28156
	K-RED2X†	**1563.1**	**1236.4**	**2760.5**	**2878**	**29834**
Kyber-512	Montgomery‡ [KS25]	6039.5	8336.0	8090.9	2902	23208
	Signed LUT† [KS25]	5274.4	7371.4	6739.8	2761	79010
	Signed LUT‡ [KS25]	5237.5	6390.4	6682.4	4297	77306
	NTT-friendly‡ (This work)	**5231.1**	**6381.2**	**6670.0**	**2761**	**21988**
LightSaber	Barrett† [HKS25]	9563.6	13503.3	14215.4	11262	62956
	NTT-friendly† (This work)	7258.1	10044.9	9598.4	11262	55632
	NTT-friendly‡ (This work)	7247.7	10028.6	9575.6	12030	53700
	Highly-Optimized (This work)	**5357.4**	**7382.2**	**7496.3**	**11266**	**28144**

†: placing LUTs/zetas in flash memory, ‡: placing LUTs/zetas in stack

Accelerating Post-Quantum KEM Schemes. To evaluate the practical benefits of the proposed modular multiplication, we integrate it into representative KEM schemes and analyze its effects on execution speed, stack usage, and code size under equivalent security levels. Our evaluation covers three candidates: CTRU-Light, Kyber-512, and LightSaber. For CTRU-Light, the K-RED2X (with modulus 769) leads to faster polynomial arithmetic across all operations. In Kyber-512, adopting the NTT-friendly variant results in speedups of 0.8%, 13.4%, and 1.0% for KeyGen, Encaps, and Decaps, respectively, while reducing the code size by 72.2% compared to the LUT-based implementation. When compared with the Montgomery-speed variant in [KS25], our version further improves these three stages by 13%, 23%, and 18%, with additional reductions of 5% in both stack usage and code size. For LightSaber, the same optimization delivers 24%, 26%, and 32% improvements in the three stages and reduces code size by 14.7%. Beyond modular multiplication, we observe that the AVR implementation of Saber in [HKS25] leaves room for optimization. We reimplement the base-case multiplication and SHA-3 routines entirely in AVR assembly, forming a highly optimized variant. Compared with their implementation, our version achieves substantial speedups of 44.0%, 45.3%, and 47.3% in KeyGen, Encaps, and Decaps, respectively. These results collectively demonstrate that the proposed modular multiplication not only accelerates key cryptographic primitives but also enables compact and performance-balanced KEM designs suitable for resource-constrained 8-bit microcontrollers.

6.3 Performance Results of CTRU-Light and Comparisons

6.3.1 Performance of CTRU-Light

Table 6 presents the performance of CTRU-Light for three implementation variants: the C reference (Ref.), the assembly-optimized version with SHA3 (ASM–SHA3), and a variant employing ASCON as a lightweight hash function (ASM–ASCON). The KRED-2X multiplication is used in all variants.

The ASM–SHA3 implementation delivers substantial performance gains over the reference baseline, achieving speedups of 2.36×, 1.83×, and 1.84× in KeyGen, Encaps, and Decaps, respectively. It also reduces code size by 23.8%. Despite the lack of explicit stack compression techniques, the assembly version maintains low memory consumption—below 3 KB for full stream.

Table 6: Performance of CTRU-Light on 8-bit AVR

Implementation	CTRU-Light.KeyGen		CTRU-Light.Encaps		CTRU-Light.Decaps		Code Size
	Cycles(k)	Stack(B)	Cycles(k)	Stack(B)	Cycles(k)	Stack(B)	
Ref.	3688.4	2351	2264.6	2321	5081.9	2886	50238
ASM-SHA3	1563.1	2495	1236.4	2309	2760.5	2878	29834
Improvement	(2.36×)	–	(1.83×)	–	(1.84×)	–	(−40.6%)
ASM-ASCON	1514.7	2351	1247.3	2463	2574.7	3028	26572
Improvement	(2.44×)	–	(1.82×)	–	(1.97×)	–	(−47.1%)

Substituting SHA3 with ASCON yields additional gains. The ASM–ASCON variant further reduces total execution cycles to 1514.7 kcycles and code size to 26,572 bytes relative to ASM–SHA3, achieving an overall speedup of up to 2.44× compared to the reference implementation. Although the dual invocation of ASCON-Hash in Encaps and Decaps marginally increases stack usage, the overall footprint remains within the low-kilobyte range.

Overall, these results demonstrate that CTRU-Light retains its lightweight design while delivering substantial gains in both speed and code compactness, highlighting its suitability for long-lived wireless sensor nodes and other resource-constrained platforms such as ARM Cortex-M0 and Cortex-M4.

6.3.2 Performance Comparison with Other Schemes

Table 7 compares the performance of the proposed CTRU-Light key encapsulation mechanism with Kyber-512 and LightSaber on 8-bit AVR platforms. Both reference schemes are configured at NIST security level 1 for a fair comparison. We use our own optimized AVR implementations of Kyber-512 and LightSaber, which already achieve higher speed and lower stack usage than existing baselines.

Table 7: Comparison of CTRU-Light with Other Post-Quantum KEMs and ECC-based Key-Establishment Schemes on 8-bit AVR.

KEM	Rings	B.W. (bytes)	KeyGen (kcycles/bytes)	Encaps (kcycles/bytes)	Decaps (kcycles/bytes)	Stack (bytes)	Code Size (bytes)
CTRU-Light (This work)	$\mathbb{Z}_{769}/(x^{512}+1)$	1152	**Speed:** 1514.7 **Stack:** 2351	**Speed:** 1247.3 **Stack:** 2463	**Speed:** 2574.7 **Stack:** 3028	3028	26572
Kyber-512 (This work)	$\mathbb{Z}_{3329}/(x^{256}+1)$	1568	**Speed:** 5231.1 **Stack:** 2132	**Speed:** 6381.2 **Stack:** 2206	**Speed:** 6670.0 **Stack:** 2761	2761	21988
LightSaber (This work)	$\mathbb{Z}_{2^{13}}/(x^{256}+1)$	1408	**Speed:** 5357.4 **Stack:** 10614	**Speed:** 7382.2 **Stack:** 11266	**Speed:** 7496.3 **Stack:** 11265	11266	28144
ECDH	**Curve**	**B.W.** (bytes)	**Security** (bits)	$k \cdot G$ (kcycles)	$l \cdot G$ (kcycles)	**Stack** (bytes)	**Code Size** (bytes)
[ADLHO10]	NIST B-233	58	112	13935	13935	2200	34600
[SS18]	NIST K-233	58	112	3186	4381	1410	40140
[PCLH20]	NIST P-256	64	128	20980	20980	n/a	17776
[LWG14]	Montgomery	64	112	6636	14819	556	15052
[LLP+17]	Four $\mathbb{Q}$	64	128	3007	6562	n/a	n/a
[DHH+15]	Curve25519	64	128	13900	13900	494	17710

∗ $k \cdot G$ and $l \cdot P$ mean fixed-base scalar multiplication and variable-base scalar multiplication, respectively.

Even under these optimized conditions, CTRU-Light exhibits superior efficiency. As reported in [KS25], Kyber-512 requires aggressive stack-saving techniques such as on-the-fly polynomial matrix computation and data streaming to fit on the 8-bit AVR devices.

In contrast, CTRU-Light naturally conforms to these limitations owing to its simpler structure and smaller modulus, and achieves stack usage comparable to that of Kyber without employing any stack optimization techniques. Furthermore, CTRU-Light requires only 3,028 bytes of stack in Decaps, substantially less than LightSaber. Moreover, its KeyGen and Encaps phases not only consume less memory but also execute faster and communicate fewer bytes, making CTRU-Light a compelling option for long-lived sensor nodes that store static public keys.

Importantly, the efficiency gain of CTRU-Light primarily stems from the scheme's structural design rather than the choice of hash function. As shown in Table 6, replacing SHA3 with ASCON yields only minor performance differences, indicating that the overall speedup arises from intrinsic algorithmic efficiency.

We further compare CTRU-Light with representative ECDH-based key establishment schemes on 8-bit AVR platforms. Even 128-bit secure elliptic-curve schemes such as Curve25519 and NIST P-256 require 13–20 million cycles for a single scalar multiplication. This cost is several times higher than the combined latency of CTRU-Light Encaps and Decaps. These results highlight that, despite being lattice-based, CTRU-Light achieves superior energy efficiency and speed compared to ECC on resource-constrained microcontrollers.

6.4 Hybrid KEM-MQTT Handshakes

Table 8 compares the performance of the original KEM-MQTT protocol [KS25] and our proposed Hybrid-KEM-MQTT protocol. Since the original implementation is unavailable, we re-implemented both protocols within a unified evaluation framework to ensure consistency and fairness. To minimize the overhead of symmetric encryption on constrained devices, we instantiate the authenticated encryption component with the lightweight standard Ascon-AEAD. The Kyber-512 and LightSaber implementations used here are also updated and further optimized versions from prior work, as listed in Table 5. As one of the key design considerations for implementing Hybrid-KEM–MQTT, we take the stack memory footprint of the underlying ECC scheme into careful account. In this work, we select Curve25519, whose 8-bit AVR implementation was reported in [DHH+15] (cf. Table 7). Curve25519 provides 128-bit security while offering a favorable balance among execution speed, stack memory usage, and code size compared to the other elliptic curves.

Previous research [KS25] indicated that porting full-fledged cryptographic libraries such as OpenSSL to ultra-constrained IoT devices typically requires around 16 KB of SRAM. In contrast, our work identifies only the essential cryptographic components required for Hybrid-KEM-MQTT, focusing on enabling secure communication from the publisher side while keeping memory usage minimal. The target hardware platform is the ATmega4808, which offers merely 6 KB of SRAM. The publisher's peak memory footprint occurs during the 0-RTT phase, when both static and ephemeral public keys must be packed and transmitted to the broker.

Table 8: Performance of Hybrid-KEM-MQTT on 8-bit AVR measured at 20 MHz.

Scheme	KEM-MQTT (Publisher Side, [KS25])		Hybrid-KEM-MQTT* (Publisher Side, This work)	
	Handshake Time	SRAM	Handshake Time	SRAM
CTRU-Light (This work)	10696.7 kcycles (0.53 s)	5986	52398.8 kcycles (2.62 s)	6023
Kyber-512 (This work)	32871.5 kcycles (1.64 s)	7329	74573.6 kcycles (3.73 s)	7489
LightSaber (This work)	39921.8 kcycles (2.00 s)	15909	81623.9 kcycles (4.08 s)	16069

∗ For Hybrid Implementation, we use Curve25519-based ECDH [DHH+15].

To ensure practical relevance, our stack measurements include function parameters such as the public key (pk). After the 0-RTT handshake, most stack space is reused in subsequent RTT rounds, keeping additional memory overhead negligible. Owing to its compact parameters and low memory demand during KeyGen and Encaps, CTRU-Light enables Hybrid-KEM-MQTT to operate with a total stack requirement of just 6,023 bytes—the only configuration that can be fully executed on the ATmega4808's 6 KB SRAM. This result demonstrates that lightweight post-quantum security can be realized even on 8-bit sensor-class hardware.

Although Kyber-512 and LightSaber cannot fit within 6 KB SRAM, we further evaluate their handshake latency on the larger ATmega1284P platform to assess protocol scalability. In this test, an MQTT publisher establishes a secure session with a broker and transmits an encrypted payload consisting of a 6-byte timestamp (t), a 1-byte topic (T), and 16-byte data (d), as illustrated in Figure 2. The results show that enabling hybrid mode increases handshake latency compared to the non-hybrid baseline. Nevertheless, among all evaluated schemes, CTRU-Light consistently achieves the fastest hybrid handshake (2.62 s), outperforming Kyber-512 (3.73 s) and LightSaber (4.08 s). It also delivers the best performance in the standard KEM-MQTT configuration, confirming that its efficiency advantage persists across both hybrid and non-hybrid settings.

In summary, the proposed Hybrid-KEM-MQTT demonstrates that practical post-quantum secure communication is feasible on 8-bit IoT platforms. The protocol combines the ultra-lightweight CTRU-Light KEM with Ascon-AEAD. This integration balances cryptographic robustness, low memory usage, and high efficiency. As a result, it offers a promising solution for next-generation wireless sensor networks.

7 Conclusions

In this work, we present CTRU-Light, a lightweight lattice-based key encapsulation mechanism specifically designed for resource-constrained platforms such as 8-bit AVR devices. By combining the structural simplicity of the NTRU framework with NTT-friendly parameters and lightweight symmetric primitives, CTRU-Light achieves both compactness and efficiency while maintaining strong post-quantum security guarantees. We further introduce an optimized modular multiplication technique tailored to the AVR instruction set, achieving lower cycle counts and reduced register usage compared to existing methods. Finally, we demonstrate the feasibility of deploying CTRU-Light in real-world communication scenarios through the Hybrid KEM-MQTT protocol, providing the first empirical system-level evaluation of post-quantum cryptography on extremely low-end IoT devices. Overall, our work bridges the gap between theoretical PQC design and practical deployment by presenting a comprehensive roadmap from algorithm design and implementation optimization to protocol integration.

Acknowledgments

We thank Vincent Hwang, Qiang Li, Hengchuan Zou, Jipeng Zhang, and Zhiyuan Zhang for their thoughtful discussions and constructive feedback. We also thank the editor and the anonymous reviewers for their helpful comments during the review process.

This research was partially supported by the Shanghai Collaborative Innovation Fund (No.XTCX-KJ-2023-54). It was also partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. RS-2025-25411243, 50%) and by the NRF grant funded by MSIT (No. 2022R1C1C1013368, 50%).

References

[AAK24] Mila Anastasova, Reza Azarderakhsh, and Mehran Mozaffari Kermani. Fully hybrid tlsv1. 3 in wolfssl on cortex-m4. In *International conference on applied cryptography and network security*, pages 376–395. Springer, 2024.

[ABC+20] Erdem Alkim, Yusuf Alper Bilgin, Mehmet Cenk, et al. Cortex-m4 optimizations for R, M-lwe schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 336–357, 2020.

[ABD+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber - algorithm specifications and supporting documentation (version 3.0). *NIST Post-Quantum Cryptography Standardization Process*, 2020.

[ACD+18] Martin R Albrecht, Benjamin R Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *International Conference on Security and Cryptography for Networks*, pages 351–367. Springer, 2018.

[ADLHO10] Diego F Aranha, Ricardo Dahab, Julio César López-Hernández, and Leonardo B Oliveira. Efficient implementation of elliptic curve cryptography in wireless sensors. *Adv. Math. Commun.*, 4(2):169–187, 2010.

[ADPS16a] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. *Cryptology ePrint Archive*, 2016.

[ADPS16b] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *USENIX 2016*, pages 327–343, 2016.

[AHKS22] Amin Abdulrahman, Vincent Hwang, Matthias J Kannwischer, and Amber Sprenkels. Faster kyber and dilithium on the cortex-m4. In *International Conference on Applied Cryptography and Network Security*, pages 853–871. Springer, 2022.

[BBB+24a] Daniel J Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J Kannwischer, Franziskus Kiefer, Thales Paiva, Prasanna Ravi, and Goutam Tamvada. Kyberslash: Exploiting secret-dependent division timings in kyber implementations. *Cryptology ePrint Archive*, 2024.

[BBB+24b] Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Marco Palumbi, and Lucas Pandolfo Perin. Enabling perk and other mpc-in-the-head signatures on resource-constrained devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):84–109, 2024.

[BBC+20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. Ntru prime: round 3. *NIST Post-Quantum Cryptography Standardization Process*, 2020.

[BBC+23] Joppe W. Bos, Olivier Bronchain, Frank Custers, Joost Renes, Denise Verbakel, and Christine van Vredendaal. Enabling frodokem on embedded devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):74–96, June 2023.

[BCD+18] Elaine Barker, Lily Chen, Richard Davis, et al. Recommendation for key-derivation methods in key-establishment schemes. *NIST Special Publication*, 800:56C, 2018.

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In *SAC 2017*, volume 10719, pages 235–260, 2017.

[BGG+16] Johannes Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. High-performance and lightweight lattice-based public-key encryption. In *Proceedings of the 2nd ACM international workshop on IoT privacy, trust, and security*, pages 2–9, 2016.

[BJL+24] Shi Bai, Hansraj Jangir, Hao Lin, Tran Ngo, Weiqiang Wen, and Jinwei Zheng. Compact encryption based on module-ntru problems. In *International Conference on Post-Quantum Cryptography*, pages 371–405. Springer, 2024.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT 2012*, volume 7237, pages 719–737, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Annual international cryptology conference*, pages 232–249. Springer, 1993.

[BRS22] Joppe W Bos, Joost Renes, and Amber Sprenkels. Dilithium for memory constrained devices. In *International Conference on Cryptology in Africa*, pages 217–235. Springer, 2022.

[CDH+19] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. Algorithm specifications and supporting documentation. *Brown University and Onboard security company, Wilmington USA*, 2019.

[CDH+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, and Tsunekazu Saito. Ntru submission. *NIST Post-Quantum Cryptography Standardization Process*, 2020.

[CGRR19] Hao Cheng, Johann GROSZSCHÄDL, Peter ROENNE, and Peter RYAN. A lightweight implementation of ntruencrypt for 8-bit avr microcontrollers. National Institute of Standards and Technology (NIST), 2019.

[CGRR20] Hao Cheng, Johann Großschädl, Peter B Rønne, and Peter YA Ryan. Lightweight post-quantum key encapsulation for 8-bit avr microcontrollers. In *International Conference on Smart Card Research and Advanced Applications*, pages 18–33. Springer, 2020.

[CGRR21] Hao Cheng, Johann Großschädl, Peter B Rønne, and Peter YA Ryan. Avrntru: Lightweight ntru-based post-quantum cryptography for 8-bit avr microcontrollers. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1272–1277. IEEE, 2021.

[CL21] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27*, pages 63–91. Springer, 2021.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT 2011*, volume 7073, pages 1–20, 2011.

[Der21] Atkins Derek. Requirements for post-quantum cryptography on embedded devices in the iot. In *Conference Name: Third PQC Standardization Conference*, 2021.

[DFGS21] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the tls 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, 2021.

[DHH+15] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography*, 77(2):493–514, 2015.

[DHK+21] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, and Gregor Seiler. Faster lattice-based kems via a generic fujisaki-okamoto transform using prefix hashing. In *ACM CCS 2021*, pages 2722–2737, 2021.

[DvW21] Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? In *ASIACRYPT 2021*, volume 13093, pages 3–32, 2021.

[EI25] Obianuju Egbuagha and Emmanuel Ikwunna. Post-quantum cryptography in practice: A literature review of protocol-level transitions and readiness. *Cryptology ePrint Archive*, 2025.

[FFW+25] Jing Fan, Bo-Yue Fang, Wei-Ze Wang, Neng-Hai Yu, Feng-Hua Li, and Long Wang. Highly efficient and light ntru-based key encapsulation mechanisms with small moduli. *Mathematics*, 13(21):3388, 2025.

[FKPY22] Pierre-Alain Fouque, Paul Kirchner, Thomas Pornin, and Yang Yu. Bat: Small and fast kem over ntru lattices. *IACR TCHES*, 2022(2):240–265, 2022.

[GKS21] Daniel Ovidiu Calin Greconici, Matthias J. Kannwischer, and Arjan Sprenkels. Compact dilithium implementations on cortex-m3 and cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–24, 2021.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *TCC 2017*, volume 10677, pages 341–371, 2017.

[HHL+23] Maurice Heymann, Hannes Hennig, Norman Lahr, Richard Petri, Julian Wälde, Thorsten Knoll, Evangelos Karatsiolis, Johannes Roth, Georg Land, Jan Philipp Thoma, Juliane Krämer, Michael Meyer, and Marcel Müller. Analysis and optimization of pqc schemes — quantumrisc work package 2 report (deliverable d2.1, version 1.0). Technical report, QuantumRISC / Fraunhofer Institute for Secure Information Technology, May 2023. Work Package 2, Deliverable 2.1, Version 1.0.

[HKS25] Vincent Hwang, YoungBeom Kim, and Seog Chung Seo. Multiplying polynomials without powerful multiplication instructions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):160–202, 2025.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, volume 1423, pages 267–288, 1998.

[HRSS17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 232–252. Springer, Cham, September 2017.

[HZZ+22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray CC Cheung, Çetin Kaya Koç, and Donglong Chen. Improved plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022.

[JZC+18] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In *CRYPTO 2018*, volume 10993, pages 96–125, 2018.

[JZM19] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In *PQCrypto 2019*, volume 11505, pages 227–248, 2019.

[KGK+25] Suparna Kundu, Archisman Ghosh, Angshuman Karmakar, Shreyas Sen, and Ingrid Verbauwhede. Rudraksh: A compact and lightweight post-quantum key-encapsulation mechanism. *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, 2025(2):647–680, 2025.

[KS25] YoungBeom Kim and Seog Chung Seo. An optimized instantiation of post-quantum mqtt protocol on 8-bit avr sensor nodes. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, pages 248–266, 2025.

[LFZZ24a] Zhichuang Liang, Boyue Fang, Jieyu Zheng, and Yunlei Zhao. Compact and efficient kems over ntru lattices. *Computer Standards & Interfaces*, 89:103828, 2024.

[LFZZ24b] Zhichuang Liang, Boyue Fang, Jieyu Zheng, and Yunlei Zhao. Compact and efficient kems over NTRU lattices. *Computer Standards & Interfaces*, 89:103828, 2024.

[LLP+17] Zhe Liu, Patrick Longa, Geovandro CCF Pereira, Oscar Reparaz, and Hwajeong Seo. Four on embedded devices with strong countermeasures against side-channel attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 665–686. Springer, 2017.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, volume 6110, pages 1–23, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: truly fast NTRU using NTT. *IACR TCHES*, 2019(3):180–201, 2019.

[LWG14] Zhe Liu, Erich Wenger, and Johann Großschädl. Mote-ecc: Energy-scalable elliptic curve cryptography for wireless sensor networks. In *International Conference on Applied Cryptography and Network Security*, pages 361–379. Springer, 2014.

[MAB+20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, and Olivier Blazy. Hamming quasi-cyclic (hqc): Third round version. *NIST Post-Quantum Cryptography Standardization Process*, 2020.

[Man16] AVR Instruction Set Manual. Atmel. *San Jose, CA, USA*, 2016.

[MR08] Daniele Micciancio and Oded Regev. Post-quantum cryptography, chapter lattice-based cryptography. *Computing*, 85(1-2):105–125, 2008.

[NIS24a] NIST. Federal information processing standard (fips) 203: Module-lattice-based key-encapsulation mechanism (ml-kem). https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, 2024.

[NIS24b] NIST. Federal information processing standard (fips) 204: Module-lattice-based digital signature algorithm (ml-dsa). https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, 2024.

[NIS24c] NIST. Federal information processing standard (fips) 205: Stateless hash-based digital signature algorithm (slh-dsa). https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, 2024.

[NIS24d] NIST. Federal information processing standard (fips) 206: Fast fourier transform over ntru-lattice-based digital signature algorithm (fn-dsa) – under development. https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, 2024.

[PCLH20] Dong-won Park, Nam Su Chang, Sangyub Lee, and Seokhie Hong. Fast implementation of nist p-256 elliptic curve cryptography on 8-bit avr processor. *Applied Sciences*, 10(24):8816, 2020.

[PG13] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *International Conference on Selected Areas in Cryptography*, pages 68–85. Springer, 2013.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[Saa19] Markku-Juhani O. Saarinen. Exploring nist lwc/pqc synergy with r5sneik: How sneik 1.1 algorithms were designed to support round5. In *Second PQC Standardization Conference*, USA, 2019. NIST / PQC Standardization. Variant of Round5 using SNEIK 1.1 primitives; discusses implementation on Cortex-M4.

[Sei18] Gregor Seiler. Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography. *Cryptology ePrint Archive*, 2018.

[SLP+17] Hwajeong Seo, Zhe Liu, Taehwan Park, Hyeokchan Kwon, Sokjoon Lee, and Howon Kim. Secure number theoretic transform and speed record for ring-lwe encryption on embedded processors. In *International Conference on Information Security and Cryptology*, pages 175–188. Springer, 2017.

[SLP+18] Hwajeong Seo, Zhe Liu, Taehwan Park, Hyeokchan Kwon, Sokjoon Lee, and Howon Kim. Secure number theoretic transform and speed record for ring-lwe encryption on embedded processors. In *Information Security and Cryptology – ICISC 2017: 20th International Conference, Seoul, South Korea, November 29–December 1, 2017, Revised Selected Papers*, volume 10779 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2018.

[SS18] Seog Chung Seo and Hwajeong Seo. Highly efficient implementation of nist-compliant koblitz curve for 8-bit avr-based sensor nodes. *IEEE Access*, 6:67637–67652, 2018.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum tls without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1461–1480, 2020.

[SSW21] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum kemtls with pre-distributed public keys. In *European Symposium on Research in Computer Security*, pages 3–22. Springer, 2021.

[ST16] S. Santesson and H. Tschofenig. Rfc 7924: Transport layer security (tls) cached information extension, 2016.

[TTM+23] Meltem Sonmez Turan, Meltem Sonmez Turan, Kerry McKay, Donghoon Chang, Lawrence E Bassham, Jinkeon Kang, Noah D Waller, John M Kelsey, and Deukjo Hong. Status report on the final round of the nist lightweight cryptography standardization process. US Department of Commerce, National Institute of Standards and Technology, 2023.

[YMÖS21] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1020–1025. IEEE, 2021.

[ZFY23] Jiang Zhang, Dengguo Feng, and Di Yan. NEV: faster and smaller NTRU encryption using vector decoding. In *ASIACRYPT 2023*, pages 157–189, 2023.