

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

CHES, October 15, 2026

Yifan Dong, YoungBeom Kim, Jieyu Zheng, Zhichuang Liang, Boyue Fang,
Seog Chung Seo, Maire O'Neill and Yunlei Zhao



**QUEEN'S
UNIVERSITY
BELFAST**

CHES, October 15, 2026

Post Quantum Standards

2024 / FIPS 203

- Key encapsulation mechanism
- Lattice-based

2024 / FIPS 205

- Digital Signature
- Hash-based

2025 / Selected

- Key encapsulation mechanism
- Lattice-based



2024 / FIPS 204

- Digital Signature
- Lattice-based

2024 / Selected

- Digital Signature
- Lattice-based

Standard algorithms have been well studied

- on high-performance platforms (AVX2, GPU, and so on).
- in security protocols such as TLS/SSL, IPsec, and DNSSEC.

PQC on IoT Devices

- ▶ Wireless sensor networks rely on 8-bit AVR nodes with only a few KB of memory. (e.g., ATmega4808: 6 KB SRAM, 48 KB flash)
 - ▶ NIST PQC standards impose heavy compute, memory, and bandwidth costs. Even highly tuned implementations often fail to run on such devices.
 - ▶ PQC evaluation centers on ARM Cortex-M4, while widely deployed 8-bit AVR platforms remain largely unexplored.
 - ▶ MQTT has no built-in cryptographic protection, and hybrid (ECDH + PQ KEM) security adds further memory and bandwidth overhead.
-

Challenges on Resource-constrained Environments

1

Scheme-level

- Standard schemes employ large moduli and high-dimensional lattices → **resource-intensive**.
 - Implementations of standards often **require aggressive stack** [BBB+24b, BBC+23, HHL+23, BRS22].
-

2

Implementation-level

- Narrow register widths & lack of support for advanced instructions.
 - Existing implementations incur **extra flash consumption** [SLP+18, KS25, HKS25].
-

3

Protocol-level

- post-quantum signatures are too heavy.
 - Hybrid approaches for incurs greater resource overhead.
-

- ## Our Focus: 8-bit AVR Microcontrollers

representative platform for low-power embedded systems and lightweight cryptography evaluation

instruction set and addressing modes

- [illegible]

Our Work

1

Lightweight PQC KEM Design

- CTRU-Light: An NTRU-based PQ KEM for 8-bit AVR sensor nodes.
 - **Small moduli, low dimensions**
 - **NTT-friendly parameters.**

2

Optimized AVR Implementation

- NTT-friendly modular multiplication for 8-bit AVR.
- **High speed** and **Low memory** implementation of CTRU-Light.
- Further improvements to Kyber and Saber.

3

Hybrid KEM-MQTT Feasibility and Empirical Analysis

- Combines ECDH + PQ KEMs (**CTRU-Light, Kyber, and Saber**) with Ascon-AEAD.
- Certificate-less operation and pre-distributed keys

Scheme-level



Implementation-level



Protocol-level

The Lightweight PQC KEM

CTRU-Light.PKE: CPA-secure NTRU encryption with scalable E_8 lattice code

$$\mathcal{R}_q = \mathbb{Z}_{769}[x]/(x^{512} + 1)$$

KeyGen

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

Encrypt

- 1: $r \leftarrow \Psi_2$
- 2: $\sigma := hr$
- 3: $c := \left\lfloor \frac{q_2}{q} (\sigma + (1 + x^{256}) \cdot \text{PolyEncode}(m)) \right\rfloor \bmod q_2$
- 4: **return** c

Decrypt

- 1: $\tilde{m} := \lfloor \frac{q}{q_2} c \rfloor \cdot f \bmod \pm q$
- 2: $m := \text{PolyDecode}(\tilde{m})$
- 3: **return** m

CTRU-Light.KEM: CCA-secure via prefix hashing FO transform [DHK+21]

- Use **implicit rejection**
- Hash a **small (e.g.32 byte), unpredictable part of the public key** into seed and shared key.
- Use the lightweight cryptography standard **ASCON** for all hashes and XOF

The Lightweight PQC KEM

- **Design Rational:** Simple, Compact and Efficient

1

Structural Simplicity

- NTRU-based Design
- Avoids duplicate public polynomial

2

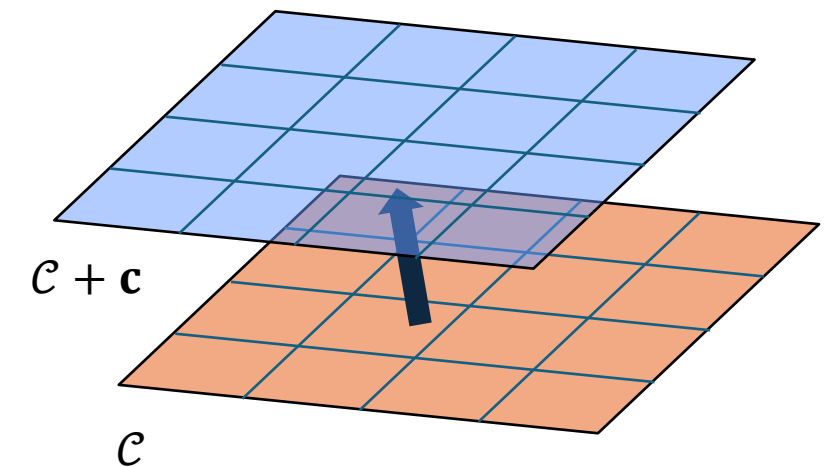
Compactness

- Small moduli and low dimensions
- Scalable E_8 lattice coding [LFZZ24a] → **low decryption failure rates**
- Vector-based encoding [ZFY23] → **reduces noise contribution**

3

Efficiency

- NTT-friendly parameters.
- Decryption directly in $\mathcal{R}_q$

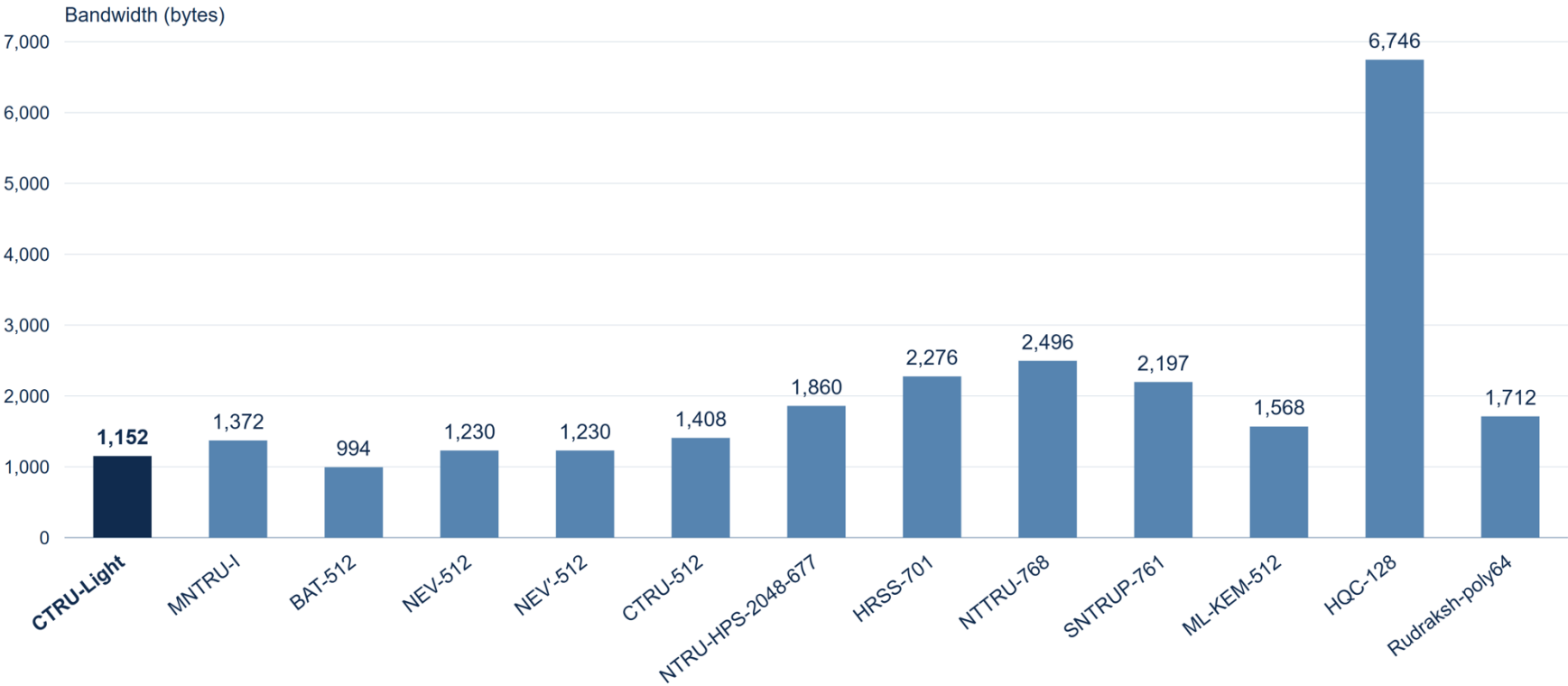


Structure of E_8

The Lightweight PQC KEM

Table 1: Recommended Parameters of CTRU-Light.

Name	n	q	q_2	(Ψ_1, Ψ_2)	$ K $	$ sk $	$ pk $	$ ct $	B.W.	(C, Q)	δ
CTRU-Light	512	769	2^8	$(\mathcal{B}_1, \mathcal{B}_1)$	16	864	640	512	1152	(125,114)	2^{-168}



► Except for BAT, CTRU-Light achieves the smallest bandwidth

BAT offers lower, but much slower key generation limits its performance in protocols requiring ephemeral keys.

The Lightweight PQC KEM

Table: Comparison of CTRU-Light with Other Post-Quantum KEMs 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)	$\mathbb{Z}_{213}/(x^{256} + 1)$	1408	Speed: 5357.4 Stack: 10614	Speed: 7382.2 Stack: 11266	Speed: 7496.3 Stack: 11265	11266	28144

2.59× - 5.11× faster than Kyber

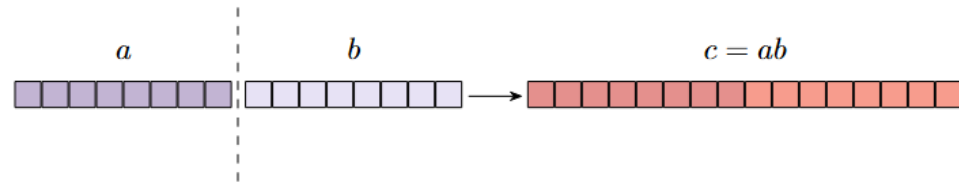
2.94× - 5.91× faster than Saber

We use our highly optimized AVR implementations of Kyber-512 and LightSaber.

- ▶ Extremely fast KeyGen, Encaps, and Decaps
- ▶ Stack usage comparable to optimized Kyber-512, **without stack-saving techniques**

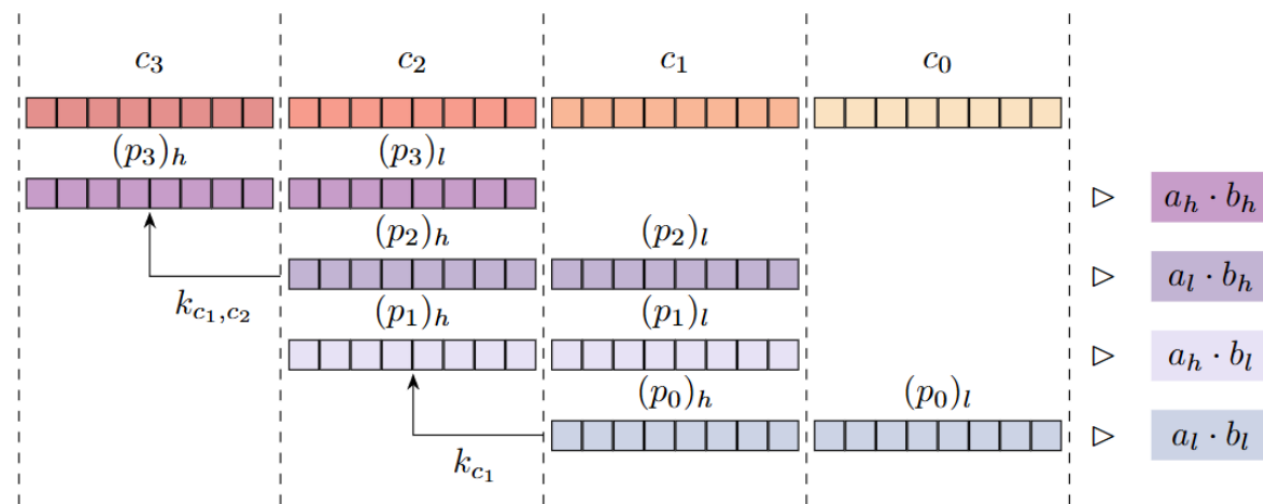
NTT-friendly Modular Multiplication on 8-bit AVR

- ▶ **8×8 long multiplication on 8-bit AVR**

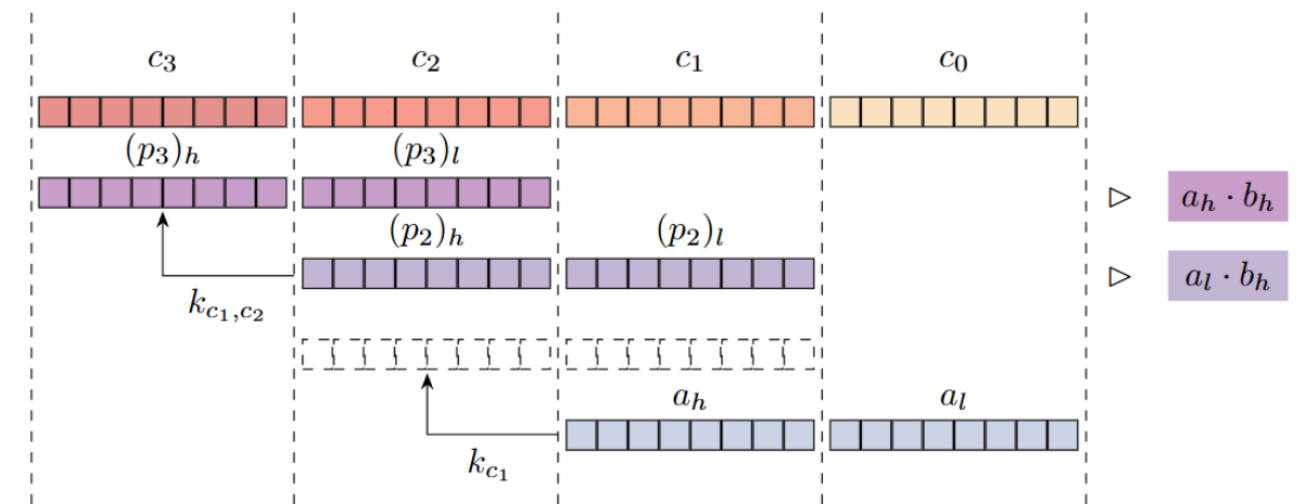


- ▶ **Simplify long multiplication via NTT-friendly factor**

▶ We focus on $b = k \times 2^8 + 1$



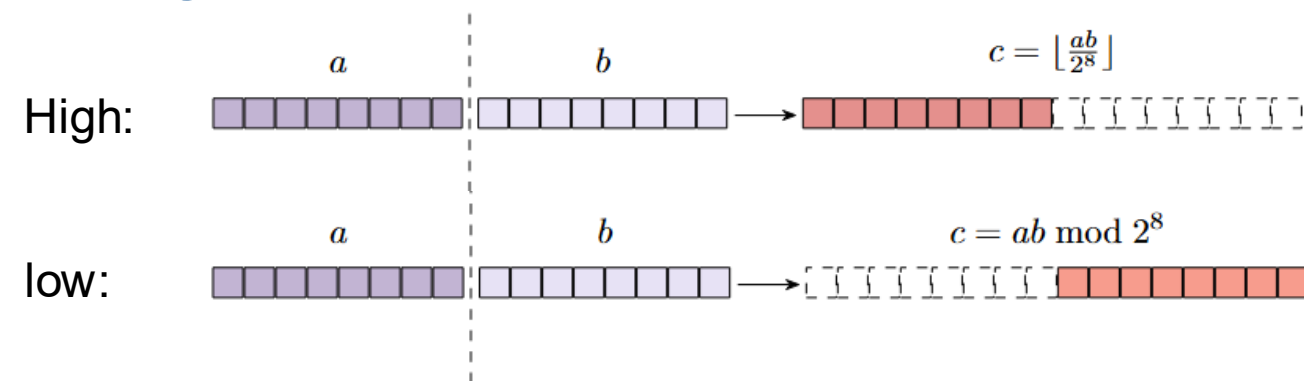
16-bit long multiplication for arbitrary b



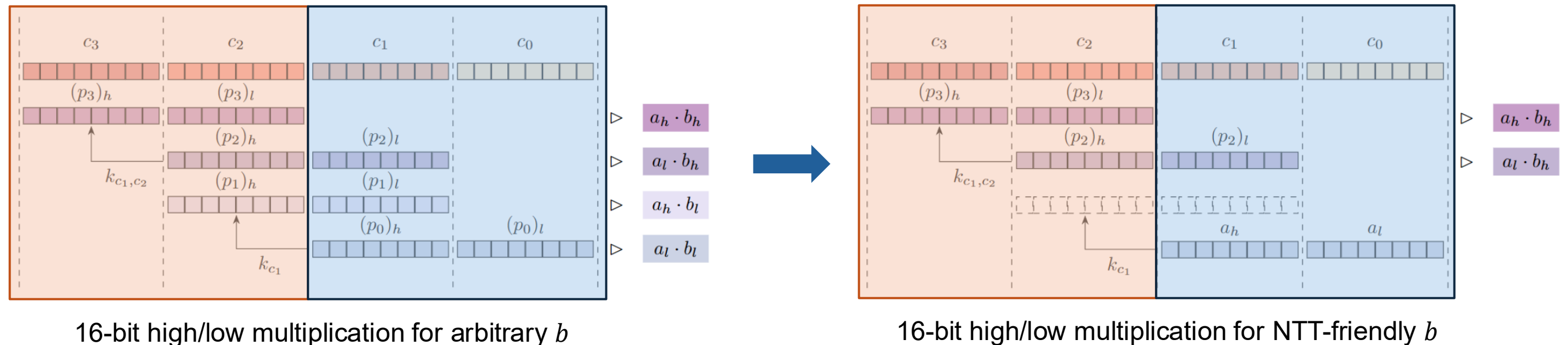
16-bit long multiplication for NTT-friendly b

NTT-friendly Modular Multiplication on 8-bit AVR

- 8×8 high/low multiplication on 8-bit AVR



- Simplify high/low multiplication via NTT-friendly factor



NTT-friendly Modular Multiplication on 8-bit AVR

► Montgomery multiplication under NTT-Friendly Moduli

Algorithm 9 Signed Montgomery multiplication [Sei18]

Input: $0 < q < \frac{R}{2}$ odd, $-\frac{R}{2} \leq a < \frac{R}{2}$, $-q \leq b \leq q$, $q' = q^{-1} \pmod R$

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: $m \leftarrow t_l q' \pmod{\pm R}$

▷ signed low multiplication, q' precomputed

3: $t_1 \leftarrow \left\lfloor \frac{mq}{R} \right\rfloor$

▷ signed high multiplication

4: $r' \leftarrow t_h - t_1$

► KRED2X multiplication under NTT-Friendly Moduli

Theorem: K-RED2X Multiplication = Montgomery Multiplication on NTT-Friendly q

Under NTT-friendly moduli admitting a 256-th root of unity, the KRED2X 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:

$$\text{KRED2X}(ab) \equiv \text{Mont}(ab) \pmod q, \quad |\text{KRED2X}(ab) - \text{Mont}(ab)| \leq (k-1)q$$

► 50 cycles → 36 cycles

NTT-friendly multiplication

► 36 cycles → 30 cycles

For small k

NTT-friendly Modular Multiplication on 8-bit AVR

► Barrett multiplication [HKS25] under NTT-Friendly Moduli

$$ab \bmod q = ab - \left\lfloor \frac{a \left\lfloor \frac{bR}{q} \right\rfloor_0}{R} \right\rfloor_1 \cdot q$$

- $\left\lfloor \frac{bR}{q} \right\rfloor_0$ is precomputed
- only the lower half of $\left\lfloor \frac{a \left\lfloor \frac{bR}{q} \right\rfloor_0}{R} \right\rfloor_1 \cdot q$ is required

► 38 cycles → 26 cycles

Low multiplication

Notions in Barrett multiplication

Name	$\llbracket \cdot \rrbracket_0$	$\llbracket \cdot \rrbracket_1$	Upper bound	Upper bound when $ a \leq \frac{R}{2}$
Standard	$\lfloor \cdot \rfloor$	$\lfloor \cdot \rfloor$	$\frac{q}{2} \left(1 + \frac{ a }{R} \right)$	$0.75q$
Floor	$\lfloor \cdot \rfloor$	$\lfloor \cdot \rfloor$	$\frac{q}{2} \left(3 + \frac{ a }{R} \right)$	$1.75q$
Half-approximate	$\lfloor \cdot \rfloor$	$b \llbracket \cdot \rrbracket$	$\frac{q}{2} \left(5 + \frac{ a }{R} \right)$	$2.75q$
Approximate	$\lfloor \cdot \rfloor$	$\llbracket \cdot \rrbracket_b$	$\frac{q}{2} \left(7 + \frac{ a }{R} \right)$	$3.75q$

- $b \llbracket xy \rrbracket = \left\lfloor \frac{x_l y_h}{\sqrt{R}} + \frac{1}{2} \right\rfloor + \left\lfloor \frac{x_h y_l}{\sqrt{R}} \right\rfloor + x_h y_h$
- $\llbracket xy \rrbracket_b = \left\lfloor \frac{x_l y_h}{\sqrt{R}} \right\rfloor + \left\lfloor \frac{x_h y_l}{\sqrt{R}} \right\rfloor + x_h y_h$

NTT-friendly Modular Multiplication on 8-bit AVR

Table: 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

- ▶ NTT-friendly multiplications **outperform** all previous approaches **without additional flash memory overhead**
- ▶ Register usage decreased

NTT-friendly Modular Multiplication on 8-bit AVR

Table: 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	15.1%- 29.3% faster
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	-	13.4%- 59.6% faster
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	-	3.3%- 89.1% faster
	$\mathbb{Z}_{7681}[x]/(x^{256} + 1)$	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	-	4.4%- 87.7% faster

AVR Assembly Implementation of CTRU-Light.

Table: 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%)

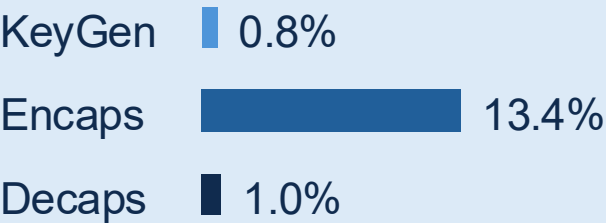
- ▶ NTT-based Polynomial Multiplication and Inversion
- ▶ NTT-friendly multiplication
- ▶ ASCON-based symmetric primitives

Highly optimized Kyber and Saber

Kyber-512

Baseline: Signed LUT, tables in Flash
Improved: NTT-friendly, twiddles on stack

Speed up:



Code size

79010 bytes → 21988 bytes
72.2% smaller

LightSaber

Baseline: Barrett, tables in Flash
Improved: NTT-friendly, twiddles on stack

Speed up:



Code size

62956 bytes → 53700 bytes
14.7% smaller

Hybrid Lightweight KEM MQTT Protocol

Publisher (AVR)

static (KEM_P) pk_P, sk_P, id_P
knows pk_B

(pk_e, sk_e) ← KEM_e.Keygen()
(ss_B, ct_B) ← KEM_B.Encaps(pk_B)

$K_B \leftarrow \text{KDF}(ss_B)$
pk_e, ct_B, AEAD_{K_B}(id_P || pk_P)

ss_B ← KEM_B.Decaps(ct_B, sk_B)
(ss_e, ct_e) ← KEM_e.Encaps(pk_e)
(ss_P, ct_P) ← KEM_P.Encaps(pk_P)

ct_e

ss_e ← KEM_e.Decaps(ct_e, sk_e)

$K_1 \leftarrow \text{KDF}(ss_e || ss_B)$
AEAD_{K₁}(ct_P)

ss_P ← KEM_P.Decaps(ct_P, sk_P)

$K_2, K'_2, K''_2, K'''_2 \leftarrow \text{KDF}(ss_e || ss_B || ss_P)$

AEAD_{K₂} (key confirmation same as TLS 1.3)

AEAD_{K'₂} (timestamp *t*)

AEAD_{K''₂} (key confirmation same as TLS 1.3)

AEAD_{K'''₂} (timestamp *t* || Topic *T* || data *d*)

KEM-MQTT[KS25]

Broker

static (KEM_B) pk_B, sk_B
knows id_P and H(pk_P)

Publisher (AVR)

static (KEM_P) pk_P, sk_P, id_P
knows pk_B
static (ECDH_s) g^{x_P}, x_P
knows g^{x_B}

(pk_u, sk_u) ← KEM_u.Keygen()
(ss_b, ct_b) ← KEM_b.Encaps(pk_B)

(g^{x_u}, x_u) ← ECDH_t.Keygen()
(ss_{ecd_hs}) ← ECDH_s.SharedSS(g^{x_B}, x_P)

$K_1 \leftarrow \text{KDF}(ss_b)$
ct_b, AEAD_{K₁}(id_P || pk_P || g^{x_P} || pk_u || g^{x_u})

ss_b ← KEM_b.Decaps(ct_b, sk_b)
(ss_u, ct_u) ← KEM_u.Encaps(pk_u)
(ss_P, ct_P) ← KEM_P.Encaps(pk_P)
(pk_v, sk_v) ← KEM_v.Keygen()

(g^{x_v}, x_v) ← ECDH_t.Keygen()
(ss_{ecd_ht}) ← ECDH_t.SharedSS(g^{x_u}, x_v)
(ss_{ecd_hs}) ← ECDH_s.SharedSS(g^{x_P}, x_B)

$K'_1 \leftarrow \text{KDF}(ss_b || ss_P || ss_{ecd_hs} || ss_u)$
ct_u, ct_P, AEAD_{K'₁}(pk_v || g^{x_v})

(ss_{ecd_ht}) ← ECDH_t.SharedSS(g^{x_v}, x_u)

ss_u ← KEM_u.Decaps(ct_u, sk_u)
ss_P ← KEM_P.Decaps(ct_P, sk_P)
(ss_v, ct_v) ← KEM_v.Encaps(pk_v)

ct_v

ss_v ← KEM_v.Decaps(ct_v, sk_v)

$K_2, K'_2, K''_2, K'''_2 \leftarrow \text{KDF}(ss_b || ss_P || ss_{ecd_hs} || ss_u || ss_v || ss_{ecd_ht})$
AEAD_{K₂} (key confirmation same as TLS 1.3)

AEAD_{K'₂} (timestamp *t*)

AEAD_{K''₂} (key confirmation same as TLS 1.3)

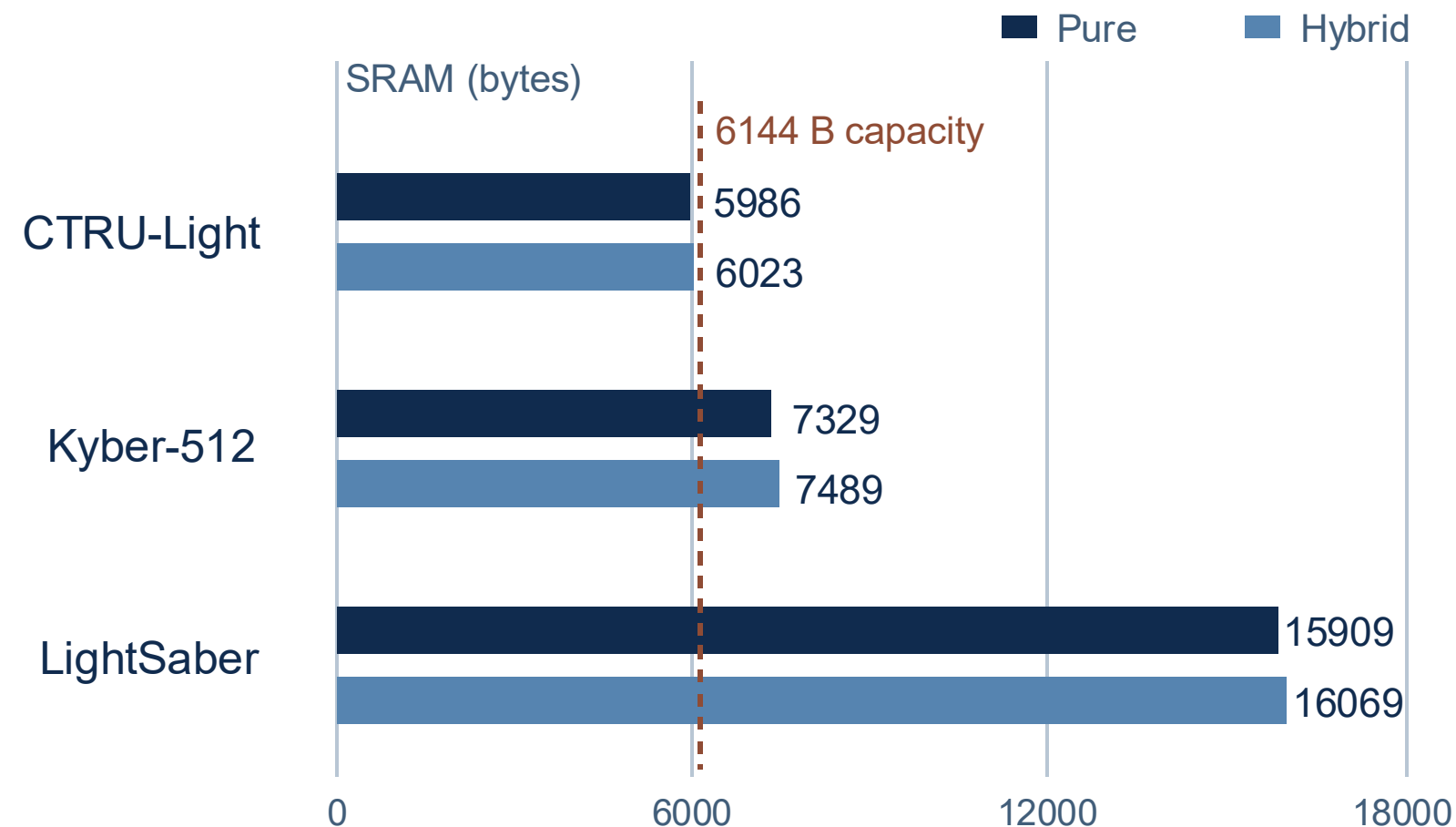
AEAD_{K'''₂} (timestamp *t* || Topic *T* || data *d*)

Hybrid KEM-MQTT

- No certificates:
Static public keys are preloaded
- Mutual authentication
through KEM instead of signatures
- Confidentiality and integrity
AEAD protection

Hybrid KEM MQTT Protocol Handshake

- **Curve25519** + CTRU-Ligh/Kyber-512/LightSaber
- AEAD is instantiated with the lightweight standard **Ascon-AEAD**
- Target hardware platform: ATmega4808 (offers **6 KB of SRAM**)

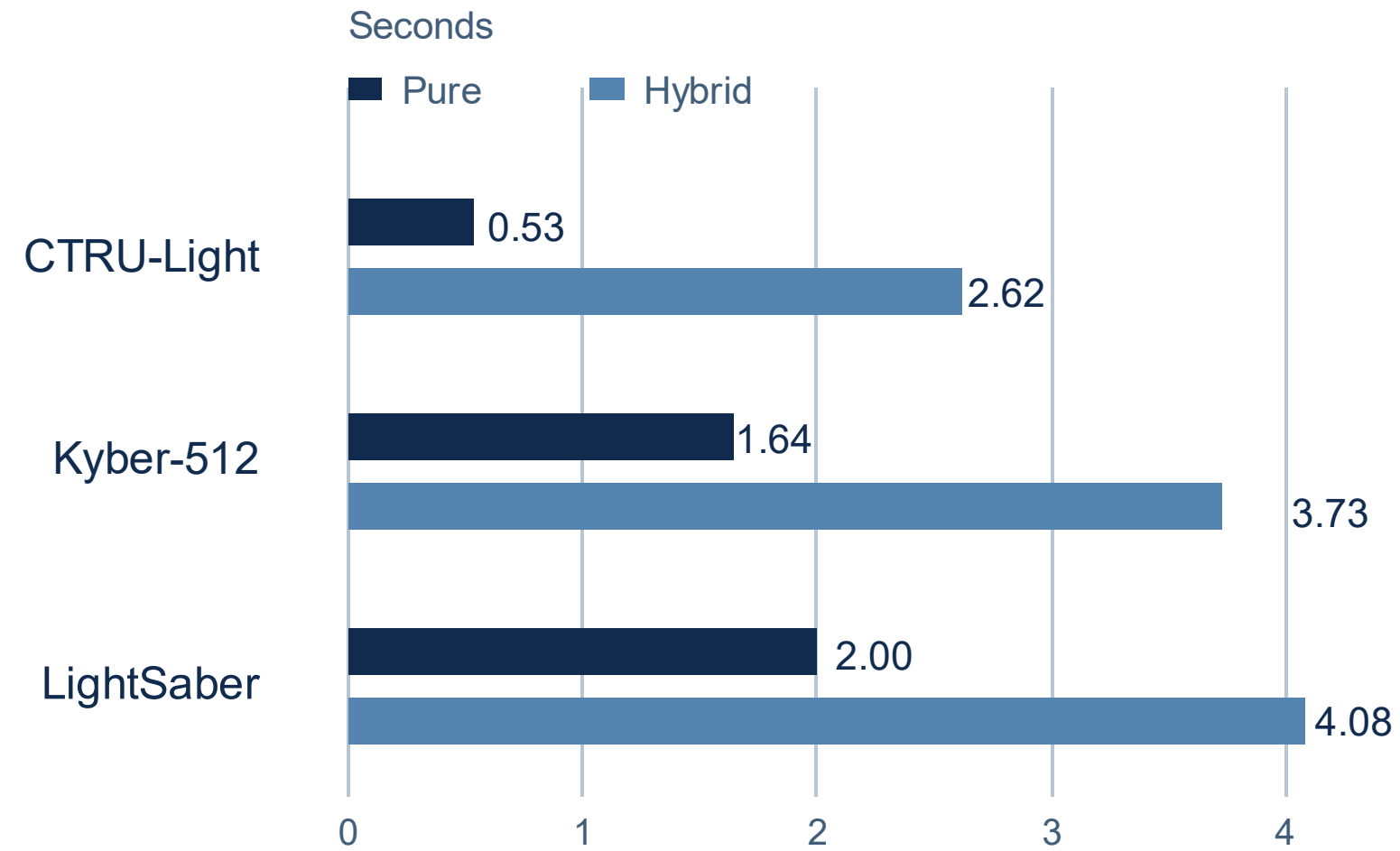


► Only +37 B
Over pure KEM-MQTT.

► 121 B margin
Fully executed on the ATmega4808.

► Only CTRU-Light fits both modes among the tested configurations.

Hybrid KEM MQTT Protocol Handshake



► **−30% / −36%**

Versus Kyber / LightSaber.

► **+2.09 s**

Hybrid overhead from original cycle counts.

► Kyber and LightSaber are evaluated on ATmega1284P.

THANKS!

Yifan Dong

yfdong24@m.fudan.edu.cn