

An Optimized Instantiation of Post-Quantum MQTT protocol on 8-bit AVR Sensor Nodes

25. 08. 28.

Kookmin University, Seoul, Republic of Korea

YoungBeom Kim and Seog Chung Seo



❖ Motivation

➤ "Quantum Computing Threat"

→ Breaking RSA, ECC, and so on.

→ Areas that need improvement: "Secure Protocols"
- For Key Establishment

➤ Researchers' responses

✓ 1st Response: Post-Quantum Cryptography (**PQC**)

✓ 2nd Response: Quantum Key Distribution (**QKD**)

✓ 3rd Response: Quantum Random Number Generator (**QRNG**)

❖ Motivation with Two Question

Q1: Is *PQC more efficient than ECDH and ECDSA?*

Q2: Is it necessary to *migrate to PQC right now?*

❖ Motivation with Two Question

Q1: *Is **PQC more efficient** than **ECDH and ECDSA**?*

- Optimized Lattice-based PQC is faster than ECDH and ECDSA
- Lower memory footprint achievable
- Good for Cortex M/A series, RISC-V, AVX2, GPU, and so on.
- Competitive handshake performance compared to TLS with ECC

Q2: *Is it necessary to **migrate to PQC right now**?*

❖ Motivation with Two Question

Q1: Is **PQC more efficient** than **ECDH and ECDSA**?

- Optimized Lattice-based PQC is faster than ECDH and ECDSA
- Lower memory footprint achievable
- Good for Cortex M/A series, RISC-V, AVX2, GPU, and so on.
- Competitive handshake performance compared to TLS with ECC

Q2: Is it necessary to **migrate to PQC right now**?

- "Store now, decrypt later" makes current data vulnerable
- Legacy ECC/DSA will eventually be broken by quantum computers
- **IoT nodes can become downgrade attack targets**
- Migration ensures long-term confidentiality and integrity

❖ Motivation with Two Question

Q1: Is **PQC more efficient** than **ECDH and ECDSA**?

- Optimized Lattice-based PQC is faster than ECDH and ECDSA
- Lower memory footprint achievable
- Good for Cortex M/A series, RISC-V, AVX2, GPU, and so on.
- Competitive handshake performance compared to TLS with ECC

Q2: Is it necessary to **migrate to PQC right now**?

- "Store now, decrypt later" makes current data vulnerable
- Legacy ECC/DSA will eventually be broken by quantum computers
- IoT nodes can become downgrade attack targets → **WE MUST ADOPT PQC!!**
- Migration ensures long-term confidentiality and integrity

❖ PQC on Embedded Device

- pqm4 project:
 - ✓ **32-bit** ARM Cortex-M4
 - ✓ <https://github.com/mupq/pqm4>

- pqm3 project
 - ✓ **32-bit** ARM Cortex-M3
 - ✓ <https://github.com/mupq/pqm3>

- Some literature for IoT Device
 - ✓ **64-bit** ARM Cortex-A
 - ✓ **32/64-bit** RISC-V
 - ✓ **16-bit** MSP430
 - ✓ **8-bit** AVR

❖ PQC on Embedded Device

➤ pqm4 project:

- ✓ **32-bit** ARM Cortex-M4
- ✓ <https://github.com/mupq/pqm4>

➤ pqm3 project

- ✓ **32-bit** ARM Cortex-M3
- ✓ <https://github.com/mupq/pqm3>

➤ Some literature for IoT Device

- ✓ **64-bit** ARM Cortex-A
- ✓ **32/64-bit** RISC-V
- ✓ **16-bit** MSP430
- ✓ **8-bit** AVR

== ALGORITHM ==

- Crystals-Kyber (FIPS 203)
- Crystals-Dilithium (FIPS 204)
- SPHINCS+ (FIPS 205)
- Falcon
- etc.

== ALGORITHM ==

- Crystals-Dilithium (FIPS 204)

❖ PQC on Embedded Device

➤ PQC-DSA on Embedded Devices (Sensor Nodes)

✓ **32-bit Cortex-M4** (No issue due to sufficient RAM)

- Huang, Junhao, et al. "Improved plantard arithmetic for lattice-based cryptography." *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.4 (2022): 614-636.

✓ **32-bit Cortex-M0** (Operable within 8 KB RAM, but extremely slow)

- Bos, Joppe W., Joost Renes, and Amber Sprenkels. "Dilithium for memory constrained devices." *International Conference on Cryptology in Africa*. Cham: Springer Nature Switzerland, 2022.

✓ **16-bit MSP430 (Dilithium: Simulation results only) → Mainly used in sensor nodes**

- Shin, Donghyun, Youngbeom Kim, and Seog Chung Seo. "Optimizing Crystals-Dilithium implementation in 16-bit MSP430 environment utilizing hardware multiplier." *ICT Express* 11.1 (2025): 59-65.

✓ **8-bit AVR (Dilithium: Simulation results only) → Mainly used in sensor nodes**

- Hwang, Vincent, YoungBeom Kim, and Seog Chung Seo. "Multiplying Polynomials without Powerful Multiplication Instructions." *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.1 (2025): 160-202.

❖ Motivation for constrained IoT Device

➤ "Quantum Computing Threat"

- Breaking RSA, ECC, and so on.
- Areas that need improvement: "Secure Protocols"
 - For Key Establishment
- 1st Response: **Post-Quantum Cryptography (PQC)**

➤ Post-Quantum-Cryptography (PQC)

- ✓ Key Encapsulation Mechanism (KEM)
 - 2 Standard (Kyber:FIPS-203 and HQC)
- ✓ Digital Signature Algorithm (DSA)
 - quite a bit bigger than KEMs
 - quite a bit slower than KEMs
 - quite a bit of extra code



Limitations on constrained devices
(Only simulation available)

Our Work



❖ 1st : Secure MQTT Protocols for Sensor Nodes

➤ MQTT

- ✓ First **PQC-based MQTT** using **KEM-TLS** for IoT Device

➤ KEM-TLS

- ✓ Removes costly **PQC-DSA** → Lightweight & practical
 - Schwabe, Peter, Douglas Stebila, and Thom Wiggers. "Post-quantum TLS without handshake signatures." *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.

❖ 2nd : Efficient Implementation of Crystals-Kyber on 8-bit AVR

➤ Signed (Small) LUT Reduction for 8-bit AVR

- ✓ *SLR* and *SSLR*

➤ Achieving "**High Speed**" + "**Low-Memory**" + "**Secure**" Implementation

- ✓ First Kyber Implementation and it outperforms ECDH (P-256) by 37.5% faster
- ✓ 3 KB stack only
- ✓ Constant-time and SCA-resistant AEAD (AES-GCM)

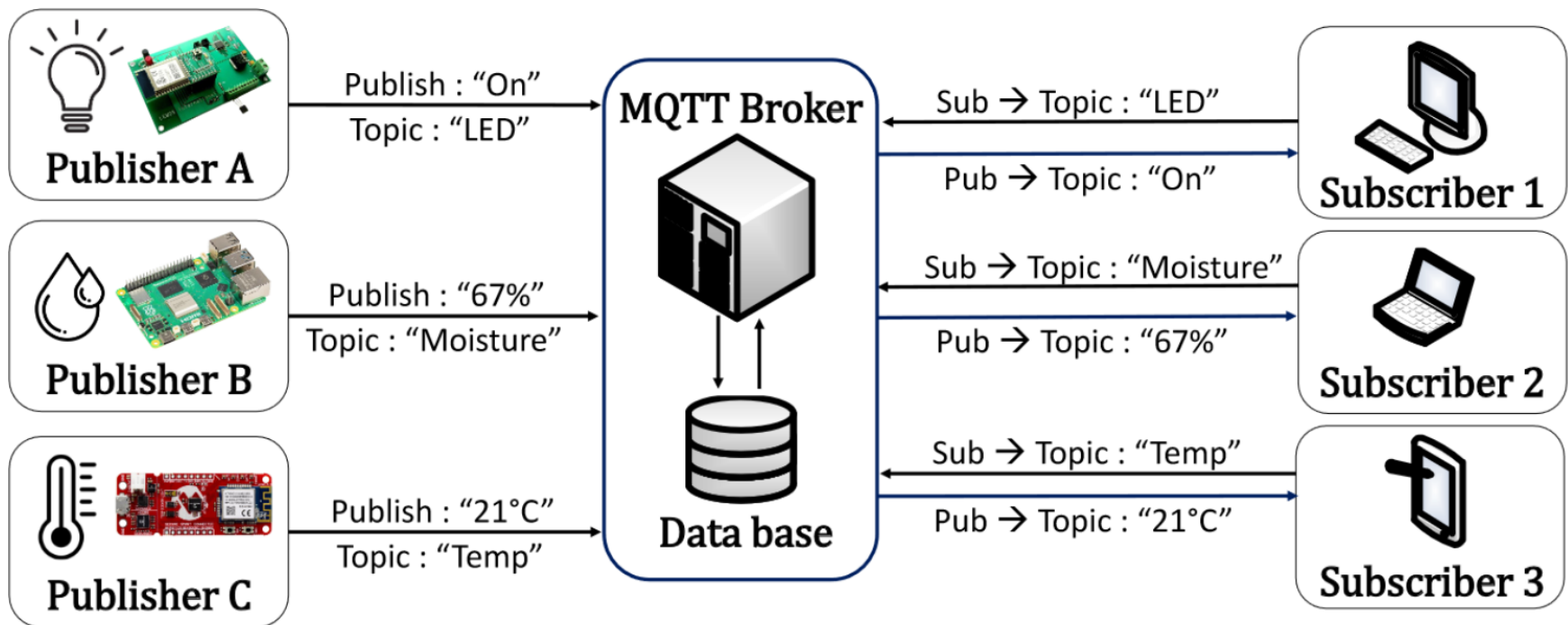
Our Work: KEM-MQTT



❖ MQTT

➤ Publish-Subscribe Protocol

- ✓ Publisher (Sensor Nodes) communicates via a Broker to publish messages.
- ✓ Subscribers receive and view the published information through the Broker.

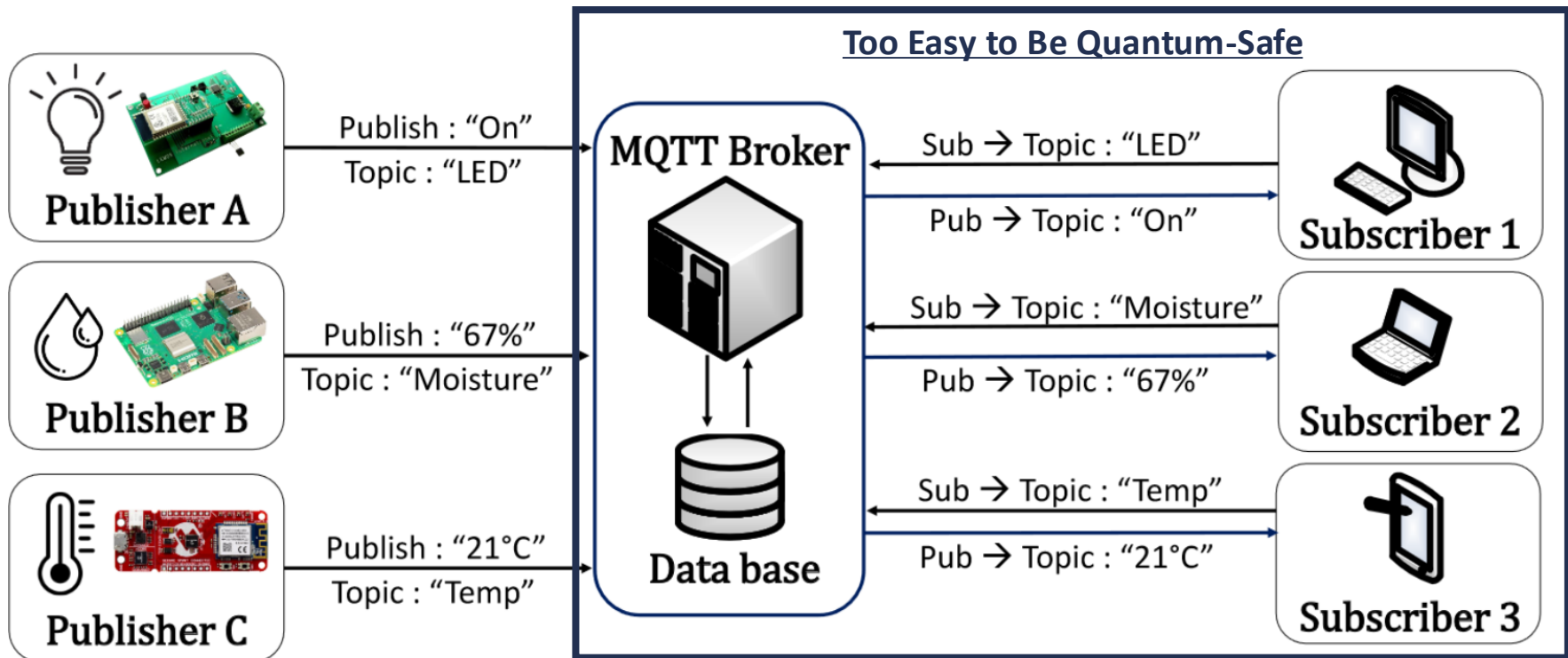


high-level overview of MQTT scenario

❖ MQTT

➤ Publish-Subscribe Protocol

- ✓ Publisher (Sensor Nodes) communicates via a Broker to publish messages.
- ✓ Subscribers receive and view the published information through the Broker.

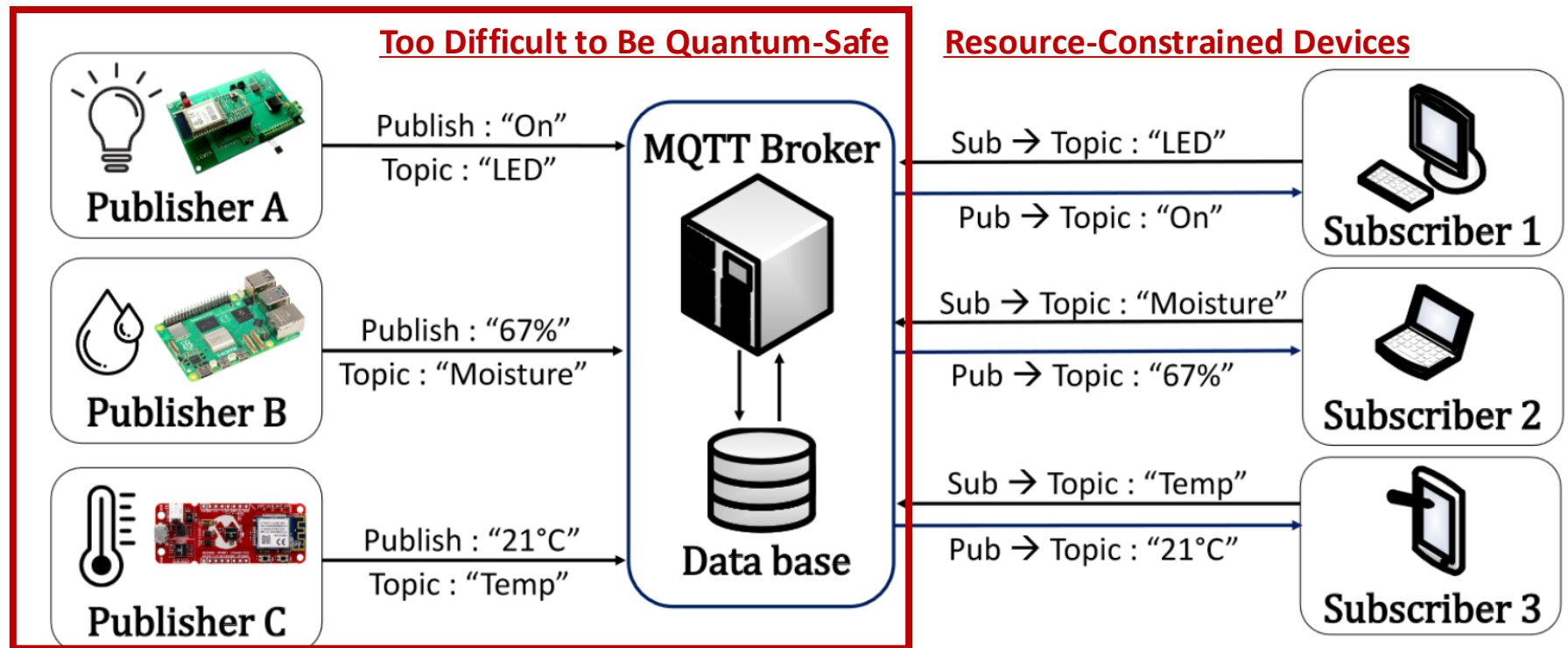


high-level overview of MQTT scenario

❖ MQTT

➤ Publish-Subscribe Protocol

- ✓ Publisher (Sensor Nodes) communicates via a Broker to publish messages.
- ✓ Subscribers receive and view the published information through the Broker.

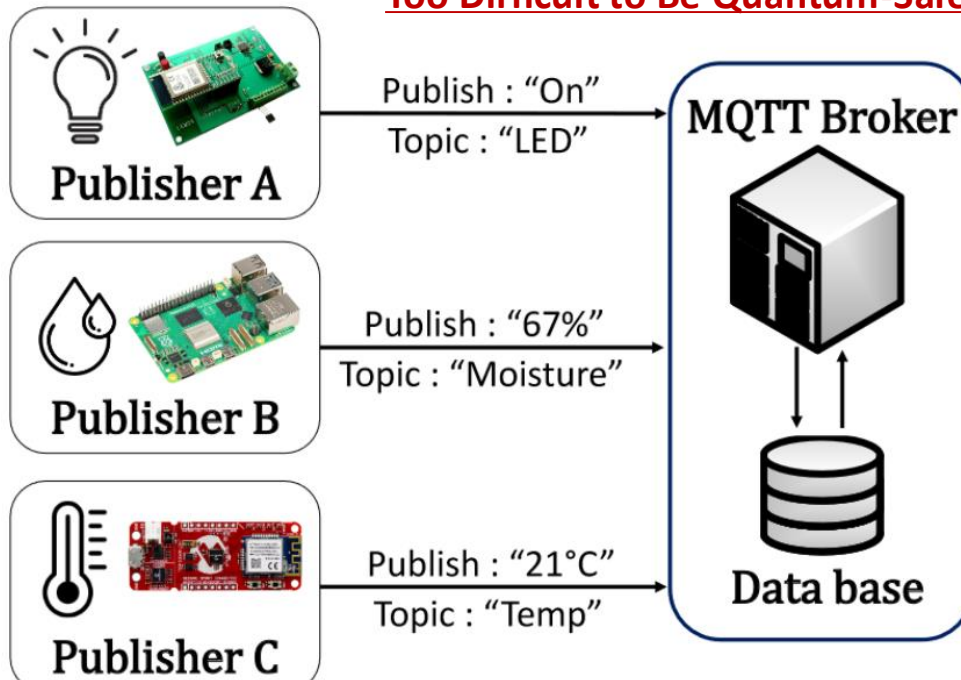


high-level overview of MQTT scenario

❖ KEM-MQTT

- Goals for Secure Communication
 - ✓ **Mutual Authentication** (Publisher & Broker)
 - ✓ **Confidentiality** and **Integrity**

Too Difficult to Be Quantum-Safe



== Single Publisher ==

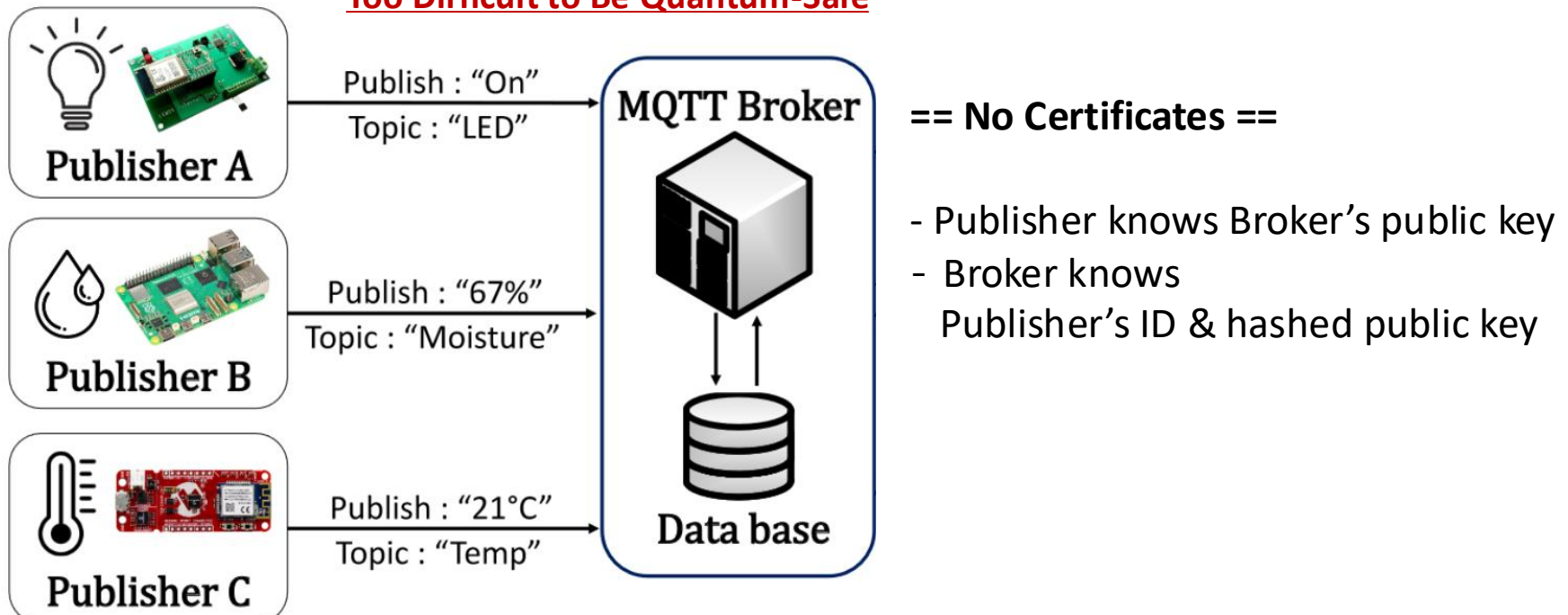
- Publisher (Sensor Node)
communicates only with the Broker
(e.g., Wildfire detection sensor)

high-level overview of MQTT scenario

❖ KEM-MQTT

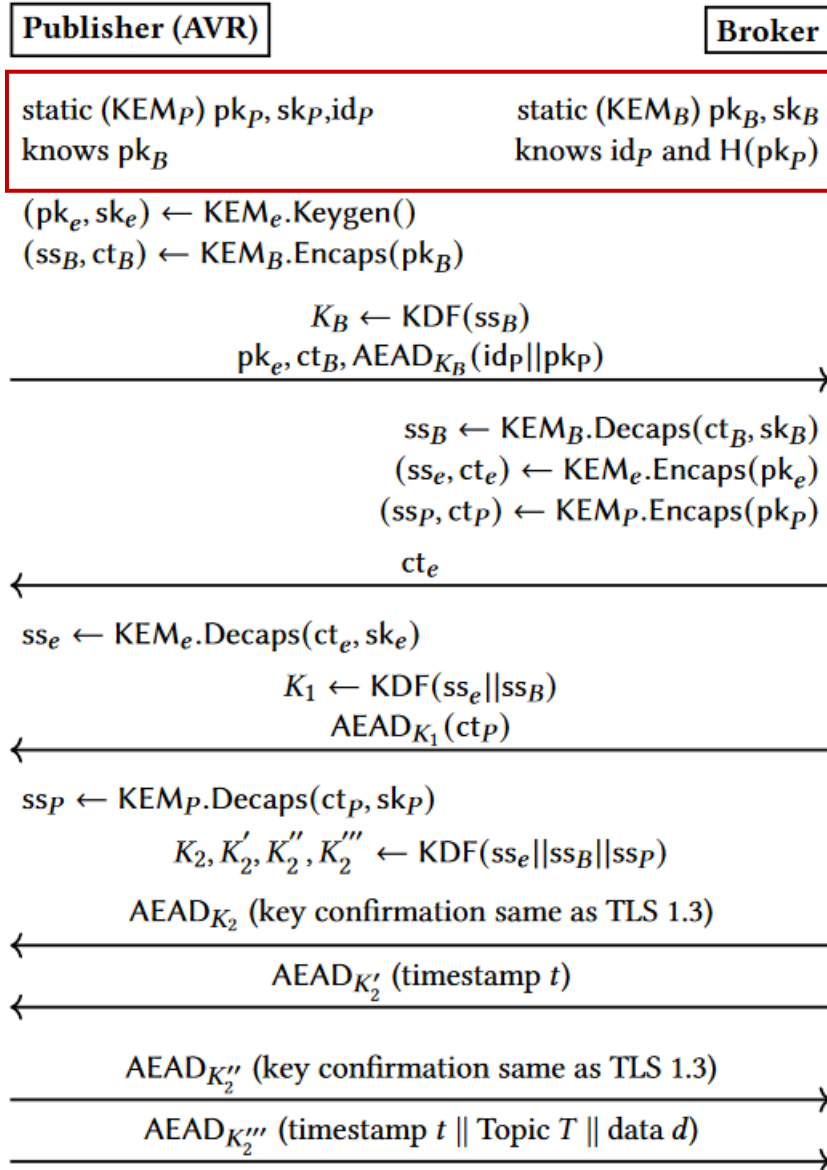
- Goals for Secure Communication
 - ✓ **Mutual Authentication** (Publisher & Broker)
 - ✓ **Confidentiality** and **Integrity**

Too Difficult to Be Quantum-Safe



high-level overview of MQTT scenario

Figure 4: Sketch of KEM-MQTT with proactive client authentication



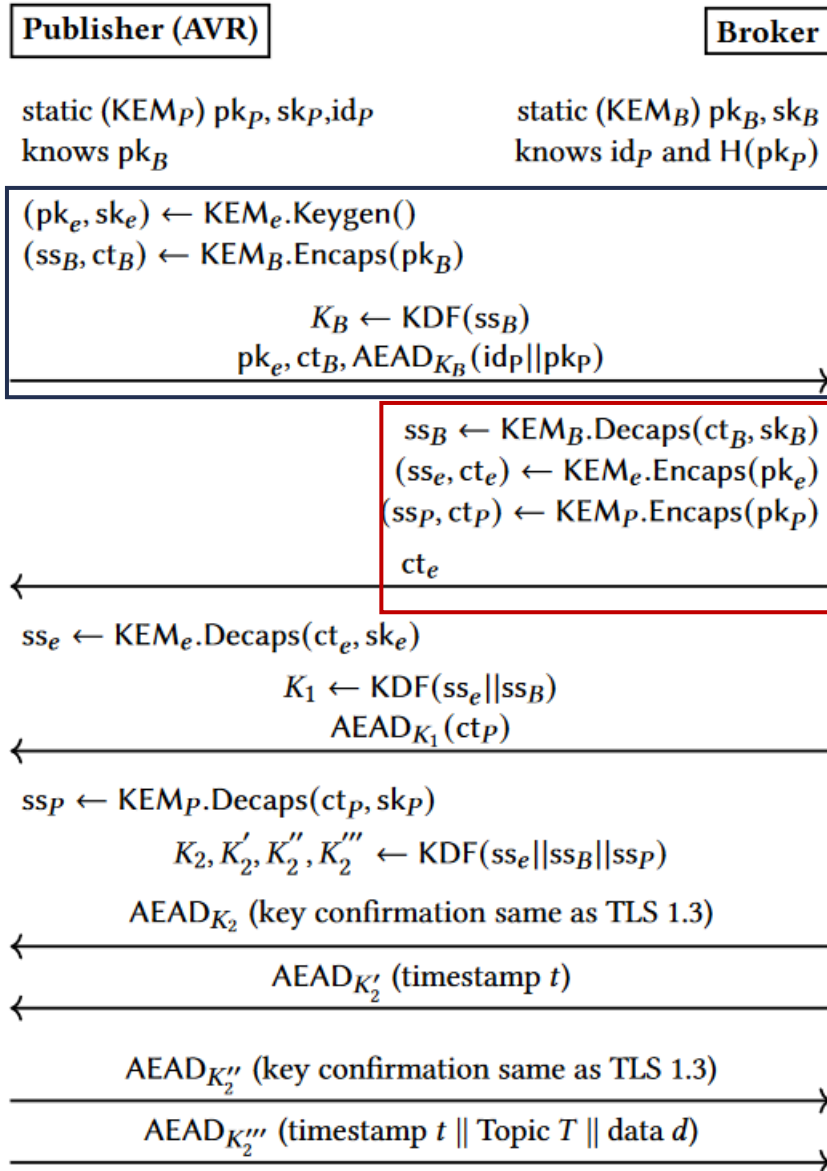
➤ Publisher (AVR)

- ✓ Store pk_P, sk_P, id_P on Flash Memory
- ✓ Store pk_B (Broker's Public key) on Flash Memory

➤ Broker (Server)

- ✓ Store pk_B, sk_B on Server DB
- ✓ Store id_P and $H(pk_P)$ on Server DB

Figure 4: Sketch of KEM-MQTT with proactive client authentication



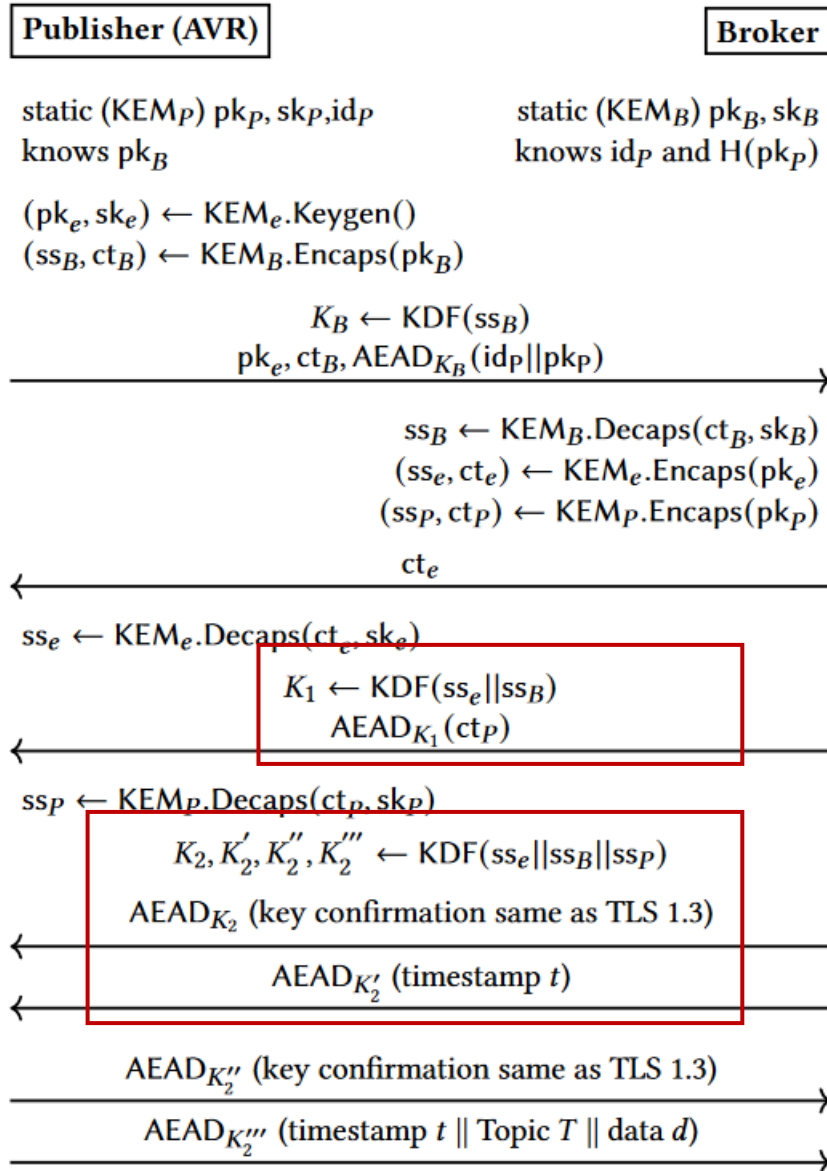
➤ Publisher (AVR)

- ✓ Generates ephemeral keys pk_e, sk_e
- ✓ Creates session key ss_b and ciphertext ct_b
- ✓ Encrypts id_p and pk_p with $(AEAD_{K_B}(id_p || pk_p))$
- ✓ Sends to Broker

➤ Broker (Server)

- ✓ Receives $pk_e, ct_b, AEAD_{K_B}(id_p || pk_p)$
- ✓ Decrypts ct_b with sk_b to obtain ss_b
- ✓ Uses ss_b to decrypt $id_p || pk_p$ (implicit authentication of publisher)
- ✓ Generates ss_e, ss_p , using pk_p, pk_e

Figure 4: Sketch of KEM-MQTT with proactive client authentication



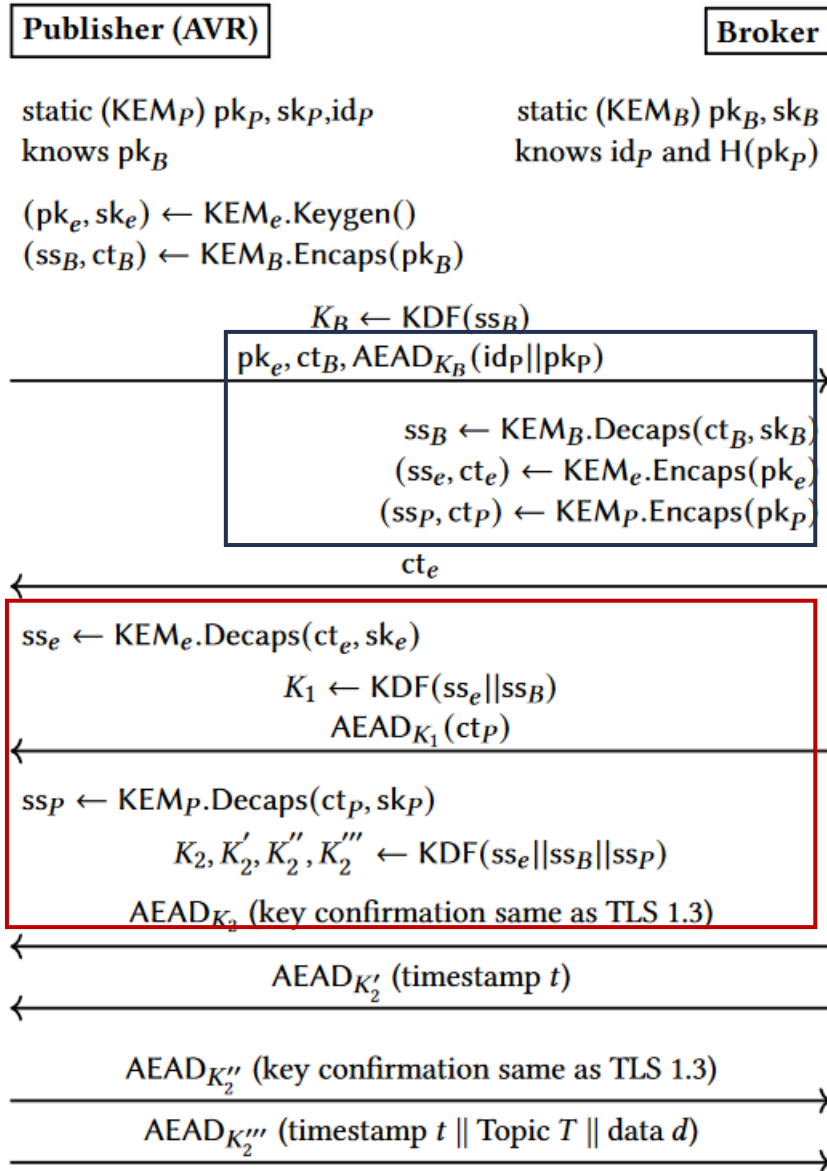
➤ Publisher (AVR)

- ✓ Generates ephemeral keys pk_e, sk_e
- ✓ Creates session key ss_b and ciphertext ct_b
- ✓ Encrypts id_p and pk_p with $(\text{AEAD}_{K_B}(id_p || pk_p))$
- ✓ Sends to Broker

➤ Broker (Server)

- ✓ Receives $pk_e, ct_b, \text{AEAD}_{K_B}(id_p || pk_p)$
- ✓ Decrypts ct_b with sk_b to obtain ss_b
- ✓ Uses ss_b to decrypt $id_p || pk_p$ (implicit authentication of publisher)
- ✓ Generates ss_e, ss_p , using pk_p, pk_e
- ✓ Derive K_1 and generate $\text{AEAD}_{K_1}(ct_p)$
- ✓ Generate key confirmation using K_2, ss_e, ss_b, ss_p
- ✓ Generate timestamp t and send all information

Figure 4: Sketch of KEM-MQTT with proactive client authentication



- **Broker** achieves **explicit authentication** if it successfully decrypts the publisher's pk_P, id_P

- **Publisher** authenticates the **Broker implicitly** via KEM, and **explicitly** via key confirmation.

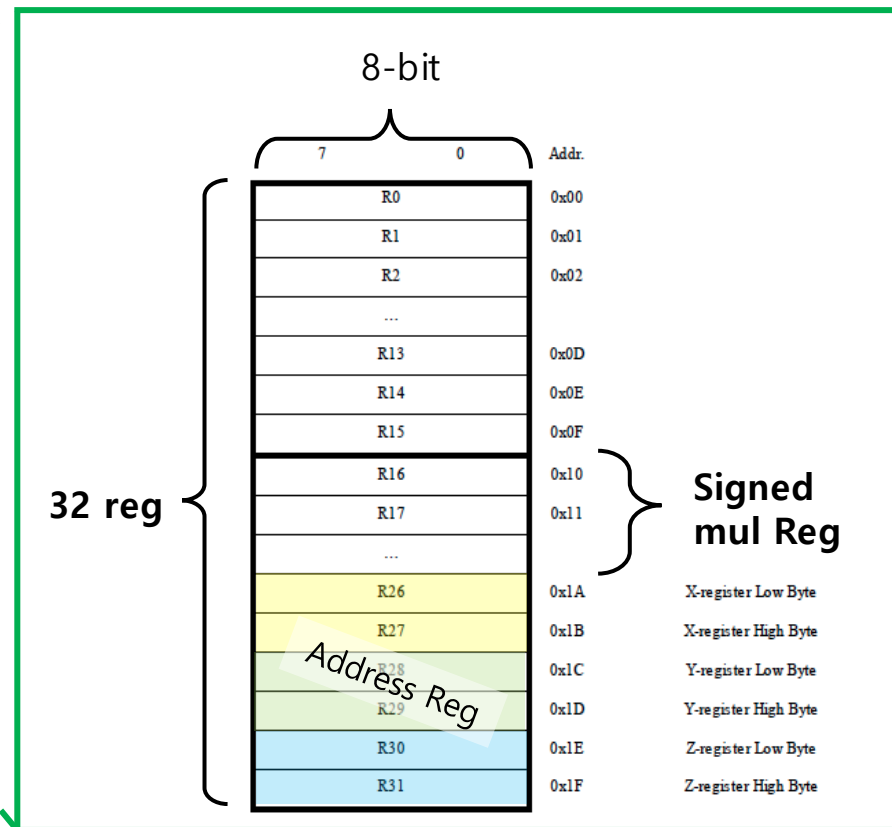
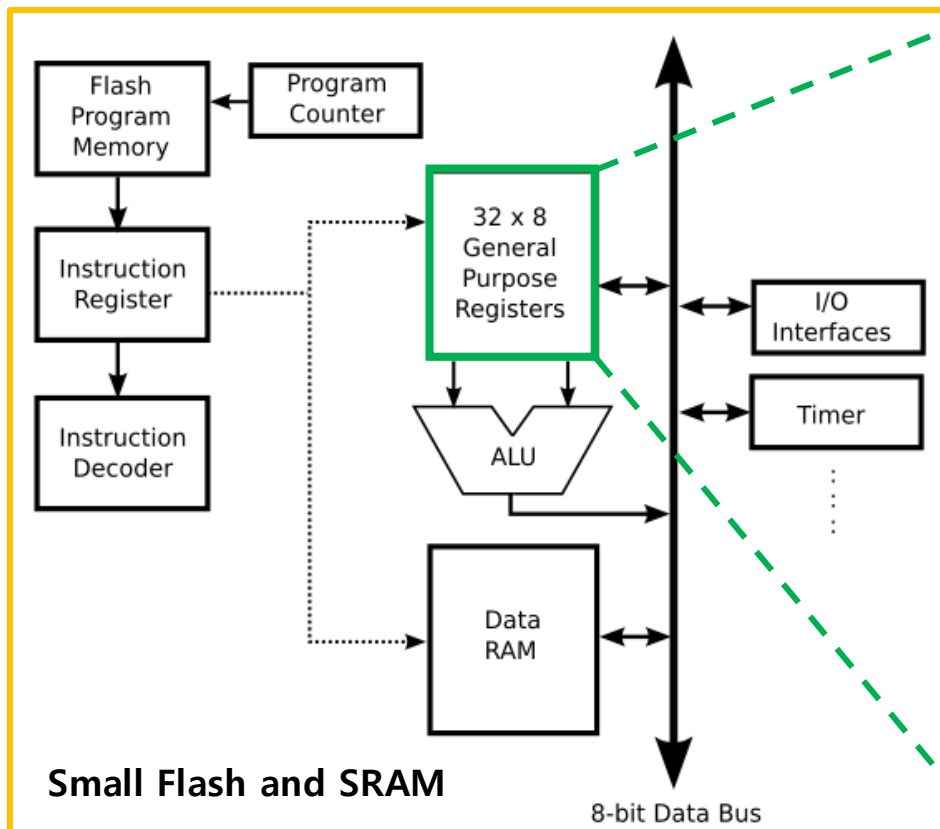
- **Confidentiality and integrity** are ensured by **AEAD**.

Our Work: Kyber on 8-bit AVR



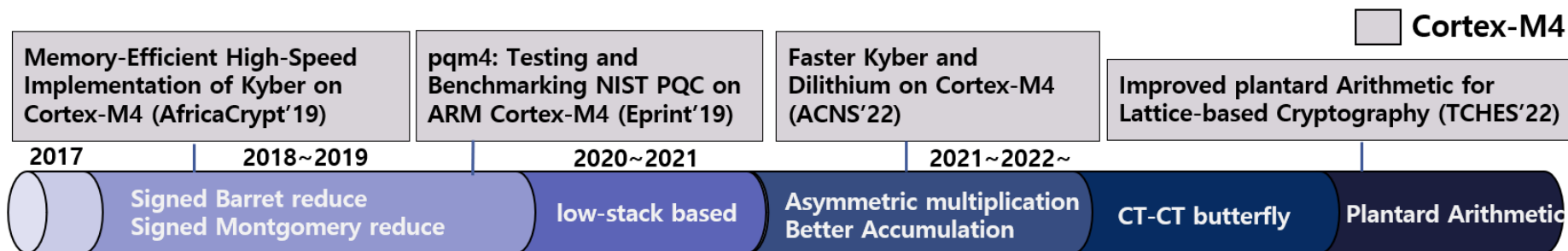
❖ 8-AVR

- 8-bit register
- Only 8-bit multiply long instruction



❖ PQC on Cortex-M4

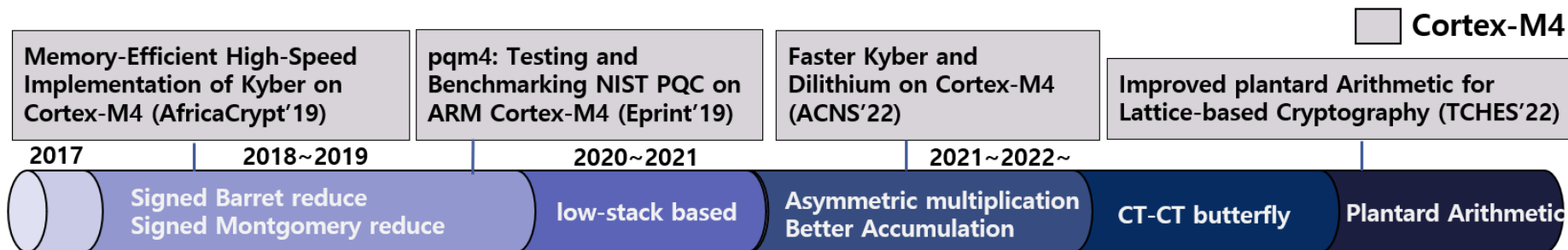
- There are Many Efficient Techniques
- Can we use these Techniques on AVR?



Kyber on Cortex-M4	Dilithium on Cortex-M4
• Streaming A and e • Signed arithmetic • Asymmetric multiplication • Better accumulation • Using Optimized Hash Function Library • Merging NTT • Plantard arithmetic NTT	• Streaming A and e • Signed arithmetic • Asymmetric multiplication • Better accumulation • Using Optimized Hash Function Library • Merging NTT • FNT NTT • low-stack strategy

❖ PQC on Cortex-M4

- There are Many Efficient Techniques
- Can we use these Techniques on AVR?



Kyber on Cortex-M4

- Streaming A and e (Suitable)
- Signed arithmetic (Suitable)
- Asymmetric multiplication (Depends on Spec)
- Better accumulation (Depends on Spec)
- Using Optimized Hash Function Library (Suitable)
- Merging NTT (Suitable)
- Plantard arithmetic NTT (Inefficient on AVR)

More Platform



Modular Arithmetic	Platform	AVR
Solinas	HW (FPGA)	✗
Plantard	Cortex-M4	✗
Montgomery	AVX2, x86/64	✗
Barrett	Cortex-A	✗
Implementation Skills	Platform	AVR
Merging Layer		✗
CT-CT butterfly		✗
Streaming Coefficient	Almost all	✓
Better Accumulation		△
Asymmetric Multiplication		△

✓: Suitable for AVR, ✗: Inefficient on AVR, △: Depends on board spec

❖ Signed LookUp-Table (LUT) Reduction (SLR)

- for Kyber's modulus $q = 3329$
- Handles **Kyber modulus ($q = 3329$)** efficiently
- Only **two LUTs** required (small footprint)
- Suitable for **8-bit AVR with 3 KB stack**
- **For 16-bit Modular Multiplication**

Algorithm Signed LUT reduction for Kyber

Input: $c = c_3 \cdot 2^{24} + c_2 \cdot 2^{16} + c_1 \cdot 2^8 + c_0$, for $c \in (-q2^{15}, q2^{15})$,
with $c_i := (c_{i[7]} \parallel c_{i[6]} \parallel \dots \parallel c_{i[0]})_{sb}$, $i \in [0, 3]$
(sb means signed binary representation),
all outputs of LUT#1 and LUT#2 in $(-\frac{q}{2}, \frac{q}{2})$

Output: $r \equiv c \bmod^{\pm} q$, where $r \in (-q, 2^{13})$

- 1: $u = (c_2 \cdot 2^{16}) \bmod^{\pm} q$ ▷ $u = \text{LUT\#1}$
- 2: $v = ((c_3[3] \parallel \dots \parallel c_3[0])_{sb} \cdot 2^{24} + (c_1[7] \parallel \dots \parallel c_1[4])_b \cdot 2^{12})) \bmod^{\pm} q$ ▷ $v = \text{LUT\#2}$
- 3: $r = u + v + (c_1[3] \parallel \dots \parallel c_1[0])_b \cdot 2^8 + c_0$
- 4: **return** r

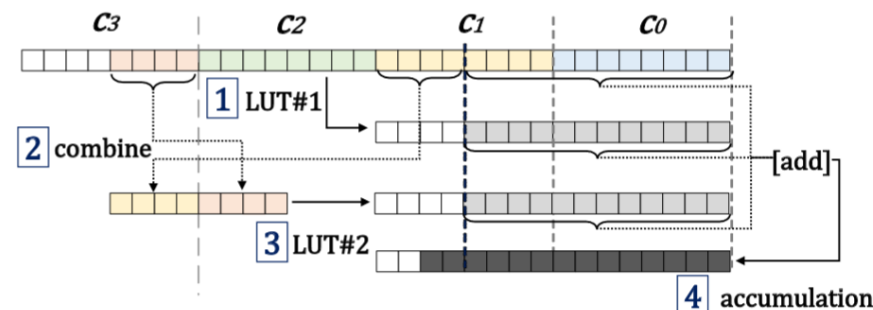


Figure : Proposed signed LUT reduction for $q = 3329$, step 1 and 3: accessing LUT; step 2: combining parts to be reduced; step 4: accumulating remainders

❖ Signed Small LUT Reduction (SSLR)

- Only **one** LUTs required (small footprint)
- **For 16-bit Modular Reduction**

Algorithm Signed Small LUT reduction for Kyber

Input: The signed integer $a = a_1 \cdot 2^8 + a_0$, for $a \in [-2^{15}, 2^{15})$

Output: $r \equiv a \bmod^{\pm} q$, where $r \in (-\frac{q}{2}, \frac{q}{2})$

1: $u = (a_1 \cdot 2^8) \bmod^{\pm} q$ ▷ LUT#3, $u \in (-\frac{q}{2}, \frac{q}{2})$

2: **return** $r = u + a_0$

Algorithm Signed small LUT reduction

Input: 16-bit $c = (c_1 || c_0)$, with $c \in [-2^{15}, 2^{15})$,
8-bit zero, LUT#3

Output: 16-bit $r = (r_1 || r_0)$, with $r \in (-\frac{q}{2}, \frac{q}{2})$

1: **mov** r30, c₁

2: **ldi** r31, hi8(LUT#3_{hi})

3: **lpm** c₁, Z

4: **ldi** r31, hi8(LUT#3_{lo})

5: **lpm** tmp, Z

6: **add** c₀, tmp

7: **add** c₁, zero

8: **return** $r_1 = c_1$ and $r_0 = c_0$

❖ Comparison

- *SLR* and *SSLR* is the fastest Modular Arithmetic on 8-bit AVR

Table 1 : Cycle counts for modular reduction on 8-bit AVR

Variant	Type	Modular Arithmetic	cycle count
32-bit reduction	Unsigned	Solinas [82]	77
	Signed	Plantard [34]	56
	Unsigned	LookUp-Table [75]	40
	Signed	Montgomery [74]	32
	Signed	LookUp-Table (<i>SLR</i>) [†]	27
	Signed	LookUp-Table (<i>SLR</i>) [‡]	23
16-bit reduction	Signed	Barrett [74]	33
	Signed	LookUp-Table (<i>SSLR</i>) [†]	11
	Signed	LookUp-Table (<i>SSLR</i>) [‡]	9

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

❖ “High Speed” Strategy

- Karatsuba multiplication
- GS & Crossed-Butterfly NTT
- Pre-hashed public key on Encapsulation

❖ “Low Memory” Strategy

- Streaming Public Matrix A
- Compact stack usage (~3 KB)

❖ “IoT-Secure” Strategy

- Constant-time Implementation
- SCA-resistant AEAD Implementation (AES-GCM)

Result



❖ Benchmarking Setup

➤ Target device

✓ ATmega4808

- 6 KB RAM and 48 KB Flash Memory
- For Kyber-512
- AVR-IOT WG DEVELOPMENT Board with 20MHz

✓ ATmega1280p (Simulation)

- 8 KB RAM and 128 KB Flash Memory
- for Kyber-768 and 1024

➤ IDE

- ✓ Microchip Studio
- ✓ avr-gcc 5.3.0
- ✓ -O3 optimization option

❖ NTT and Kyber

- Compared C NTT code, achieving **111/71/120% performance improvement** in NTT/basemul/iNTT
- Compared ECDH (P-256), **Kyber-512 is 37.5% faster**.

Table 5: Cycle counts for NTT/iNTT and single basemul

Implementation	NTT	basemul	iNTT
This work (Signed LUT [‡])	58,364 (+111%)	319 (+71%)	67,792 (+120%)
This work (Signed LUT [†])	62,204 (+98%)	328 (+67%)	72,398 (+106%)
This work (based [16])	123,363 (-)	547 (-)	149,4778 (-)

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

Table 6: Comparison of Key establishment performance between Ephemeral ECDH and Kyber on 8-bit AVR MCUs, 1,000 cc is denoted by k and s means a second

Work	Security	$k \cdot P$	$l \cdot Q$	ECDH(cc)	ECDH(s)
[79]	GF(p), 160	9,920k	10,80k	20,720k	2.81
[47]	GF(p), 160	15,100k	16,960k	32,060k	4.35
[47]	GF(p), 192	21,370k	21,370k	42,740k	5.79
[48]	GF(p), 192	3,460k	8,620k	12,080k	1.63
[31]	GF(p), 224	n/a	17,520k	24,550k [⊕]	3.33 [⊕]
[84]	GF(p), 256	n/a	25,380k	35,560k [⊕]	4.82 [⊕]
[62]	GF(p), 256	n/a	20,980k	29,400k [⊕]	3.98 [⊕]
variant	Kyber-512(s)	Kyber-768(s)	Kyber-1024(s)		
This (★◇‡)	2.49	-	-		
This	-	3.87 (★◇†)	6.02 (★†)		
This [16] (★)	4.73	7.30	11.14		

★, ◇, †, and ‡: same as Table 7, ⊕: a roughly measured

❖ Kyber

- Kyber-512/768/1024 achieves performance improvements of 82/81/80%, 101/101/99%, and 86/85/80% in Keygen, Encaps, and Decaps, respectively.

Table 7: Cycle counts (cc) and stack usage (bytes) for all security level of Kyber on 8-bit AVR MCUs. 1,000 cc is denoted by k.

Implementation	Variant		Kyber-512		Kyber-768		Kyber-1024	
			cc	stack[B]	cc	stack[B]	cc	stack[B]
This work (Signed LUT)	★ ◇ ‡	K	5,253k (+82%)	3,720	-	-	-	-
		E	6,402k (+101%)	3,808	9,694k (+101%)	4,308	14,974k (+99%)	4,828
		D	6,706k (+86%)	3,720	-	-	-	-
This work (Signed LUT)	★ †	K	5,290k (+81%)	2,220	8,608k (+81%)	2,736	13,640k (+80%)	3,256
		E	7,385k (+75%)	2,308	11,169k (+75%)	2,808	16,946k (+76%)	3,328
		D	6,763k (+85%)	2,324	10,250k (+85%)	2,824	15,773k (+80%)	3,352
This work (based [16])	★	K	9,541k	2,220	15,554k	2,736	24,433k	3,256
		E	12,879k	2,308	19,500k	2,808	29,347k	3,328
		D	12,484k	2,324	18,796k	2,824	28,377k	3,352

★: based on the stack optimize version of [16], ◇: pre-hashed pk in flash memory, †: placing LUTs in flash memory, ‡: placing LUTs in stack

❖ KEM-MQTT

➤ Handshake latency

- ✓ Publisher → Broker: 1.88 s
- ✓ Broker → Publisher: 2.44 s
- ✓ Total $\approx$ 4.32 s (excluding physical transmission)

➤ Memory Usage

- ✓ Peak Stack Usage : 5,726 (Publisher)

➤ Operation

- ✓ KEM: Kyber-512
- ✓ Key Schedule: TLS 1.3 Style
- ✓ AEAD: AES-GCM
 - Hand-written assembly
- ✓ Packet Generation

Table 8: Average handshake times in second for mutually authenticated KEM-MQTT experiments (Excluding the physical transmission time during handshake). Also, We report the maximum memory peak during the handshake.

KEM-MQTT	Publisher sent req.		Publisher recv. resp.	
	speed	stack[B]	speed	stack[B]
Kyber-512 (★ ◇ †)	13,876k 1.88s	5,726	17,957k 2.44s	5,184

★, ◇, and †: same as Table 7

Q & A

