AsiaCCS 2023 Tutorial

Securing Communications in the Post-Quantum Era

Raymond K. Zhao, CSIRO's Data61, Australia

Sara Jafarbeiki, Monash University, Australia

July 2023

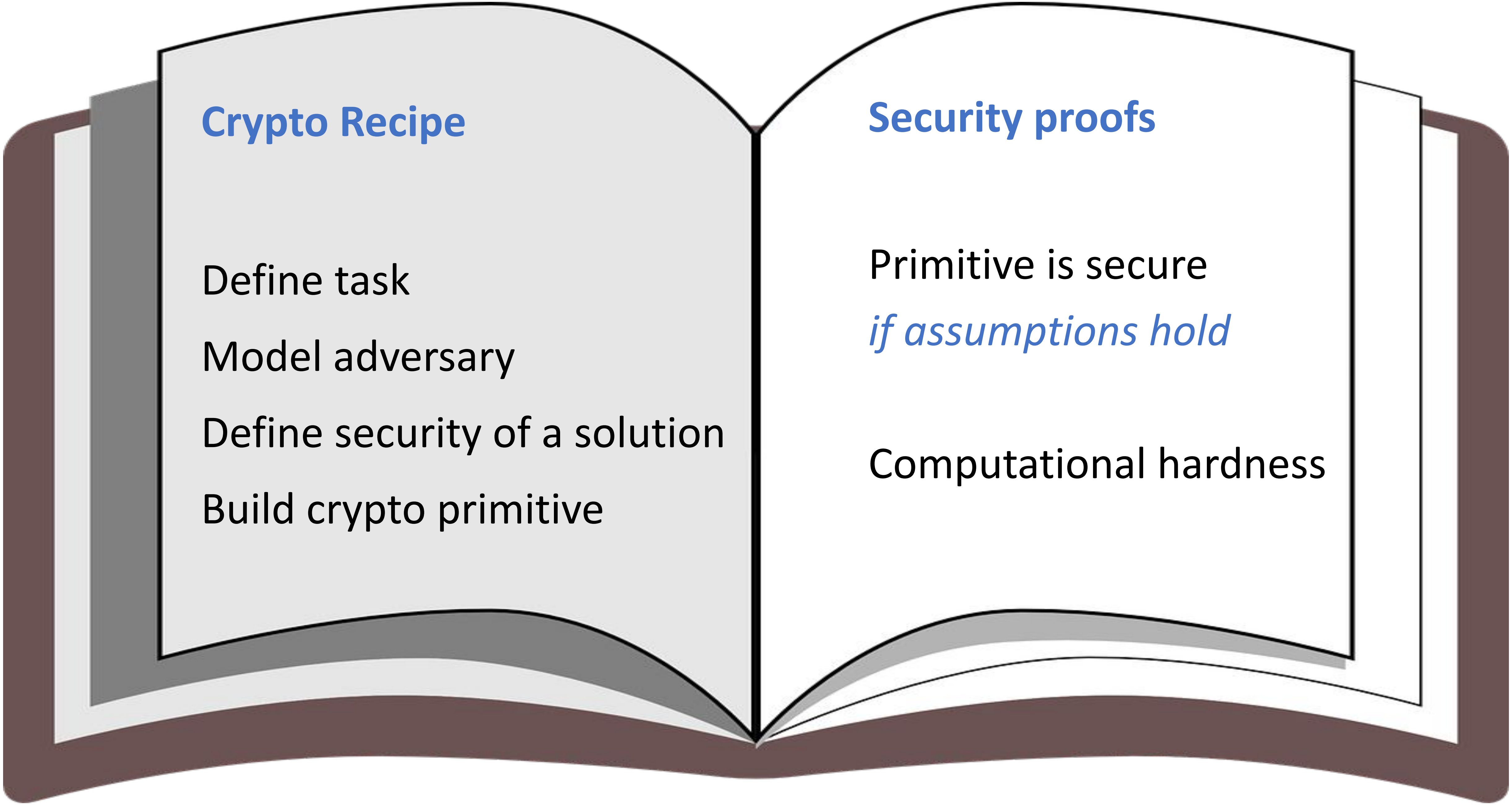
Cryptography is everywhere

Secure communication

Content protection

User authentication

and much much more…



Crypto Recipe

Define task

Model adversary

Define security of a solution

Build crypto primitive

Security proofs

Primitive is secure

if assumptions hold

Computational hardness

Hard problems and Public Key Cryptography (PKC)

- Public-key crypto inherently requires hard computational problems.
  For one: must be hard to compute the secret key from the public key.
- Issue: we don't know whether hard problems exist!
- 'Solution': conjecture that they do exist—in general, or specifically.
  Then devote scrutiny and algorithmic effort to gain confidence.

"Cryptographers seldom sleep well." –Silvio Micali

Case study:

RSA/DH are based on the hardness of Factoring/Discrete-Log variants.

How Hard, and Hard How?

- We need crypto problems to be infeasible for any attacker to solve.
- Traditionally, ‘attacker’ = classical algorithm.
- But for quantum algorithms, ‘feasible’ appears broader…

Quantum Computing

- Concept suggested by quantum physicists Paul Benioff and Richard Feynman (early 1980s)
- Exploit quantum mechanics to process information

  Use quantum bits = "**qubits**" instead of 0's and 1's

  Qubits can be in "**superposition states**": ability of quantum system to be in multiples states at the same time

  Massive parallelisation potential to vastly increase computational power beyond classical computing limit

- Computational problems that are infeasible for classical computers may become easy for quantum computers
- Can have **huge impact on cryptography**!

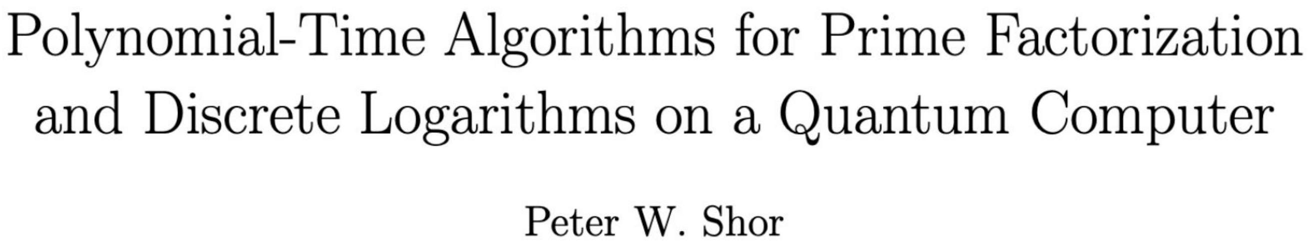
Quantum Computing Threat to Cryptography

- 1994: Shor's Algorithms – exponential speedup of QCs for breaking classical Public Key Crypto
  - Implication**: Large Scale** QC ⟶ RSA and Diffie-Hellman public-key systems become insecure!

- 1996: Grover's Algorithm – polynomial speedup of QCs for breaking Symmetric-Key Crypto
  - Currently is the best known quantum attack against AES.
  - The security of AES against quantum computers is at least ½ of the classical bit level security.

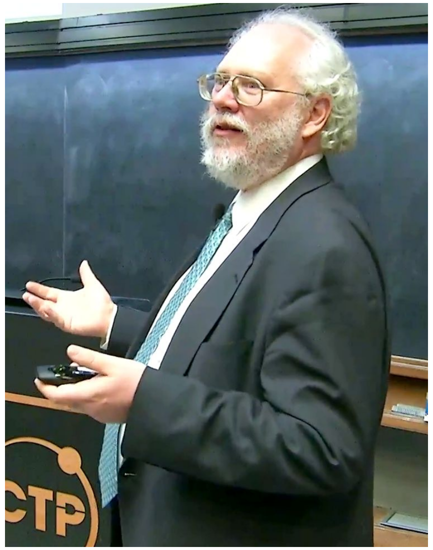
How Hard, and Hard How?

- We need crypto problems to be infeasible for any attacker to solve.
- Traditionally, ‘attacker’ = classical algorithm.
- But for quantum algorithms, ‘feasible’ appears broader:



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer

Peter W. Shor



- With a large-scale QC, Shor’s algorithm totally breaks DH, RSA, and all other widely used public-key crypto!

Post-Quantum Cryptography (PQC)

Question

Did Shor show that secure PKC is impossible against quantum computers?

Answer

No! Only that all the PKC we’ve been widely using is quantumly broken.

Post-Quantum Cryptography (PQC)

Question

Did Shor show that secure PKC is impossible against quantum computers?

Answer

No! Only that all the PKC we've been widely using is quantumly broken.

Post-Quantum Cryptography (a.k.a, 'Quantum Resistant', 'Quantum Safe', …)

Design cryptosystems that can run on (today's) classical computers,

while being secure against quantum attacks.

What’s the Rush?

- Big QCs probably won’t exist for many years, if ever—can’t we wait until they’re more imminent? No!

  - Harvesting attacks: store today’s keys/ciphertexts to break later.
  - Rewrite history: forge signatures for old keys.

    “Who controls history controls the future.”

    –George Orwell, 1984

  - Deploying new cryptography at scale takes a long time: 10+ years

    “Our ultimate goal is to provide cost effective security against a potential quantum computer.” *–NSA, 2015*

Quantum Computing Threat to Cryptography – How Far Away?

Technological Improvements in QC:

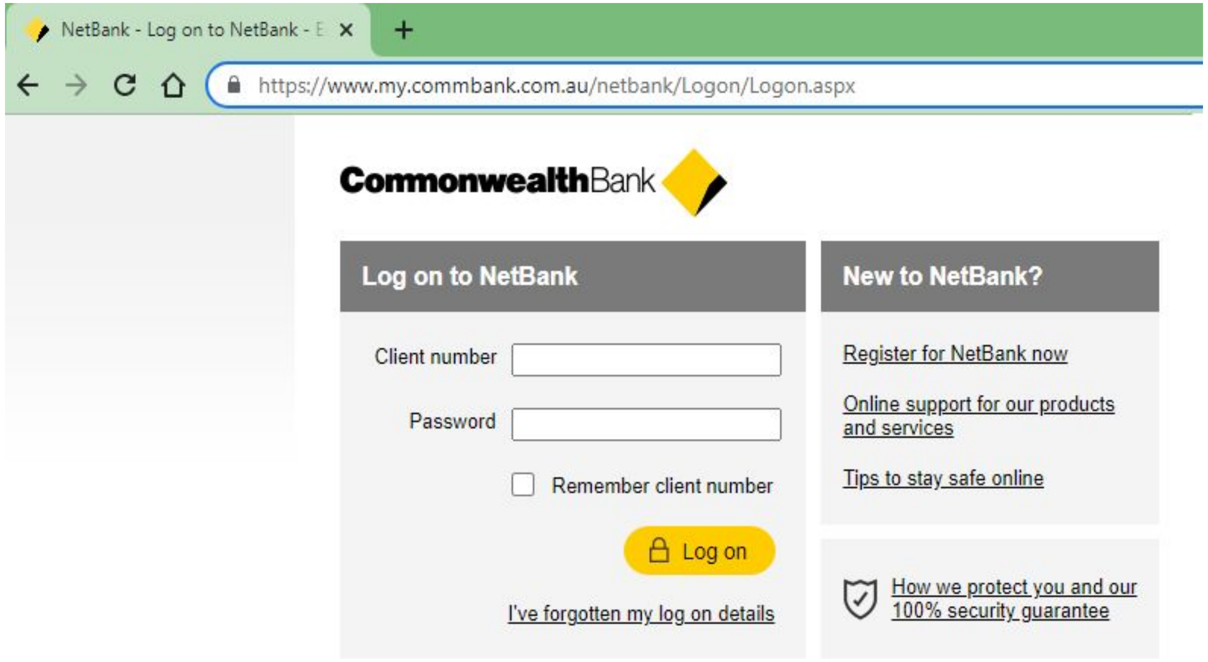
1980-82: idea proposed by Benioff / Feynman
1998: first 2-qubit quantum computer realized
2000: 7-qubit quantum computer
2006: 12-qubits
2019: 53 qubits (IBM), Google's successful quantum supremacy experiment published
2021: IBM 'Eagle' - 127 qubits
2022: IBM 'Osprey' - 433 qubits
2023 (IBM Roadmap): 1121 qubits?

IBM is aiming for 100K qubits, 2026+, https://www.ibm.com/quantum/roadmap

Concrete estimates for Elliptic Curve Discrete-Log implementation of Shor's algorithm: ~2124 qubits, ~2.3 x 10^9 quantum gates

Common Applications Utilising PKC

- ➢ Web TLS protocol
- ➢ IPSec VPN
- ➢ Database Systems
- ➢ Access Control/Authentication Systems
- ➢ Internal Systems Using Encryption
- ➢ Multi-Factor Authentication (MFA) Apps
- ➢ Operating Systems



NetBank - Log on to NetBank - E

https://www.my.commbank.com.au/netbank/Logon/Logon.aspx

CommonwealthBank

Log on to NetBank

Client number

Password

□ Remember client number

Log on

I've forgotten my log on details

New to NetBank?

Register for NetBank now

Online support for our products and services

Tips to stay safe online

How we protect you and our 100% security guarantee

PQC Migration

- ➢ The Threat ✓
- ➢ The When ✓
- ➢ The Response → Key steps to take:
  - ■ PQC Diagnosis; making an asset inventory
    - Risk assessment
    - Inventory of all cryptographic assets used in the organisation
    - Inventory of all the data handled by the organisation
  - ■ Planning the Migration
    - Urgency
    - Business process planning; cost
    - Technical planning; Cryptography Replacement, Hardware Replacement
  - ■ Executing the Migration

Post-Quantum Cryptography in the Indo-Pacific Program (PQCIP)

Stage 1

Identify and engage with Participants.

Stage 2

Assess participants understanding of PQC.

Stage 3

Identify the main PQC vulnerable systems of participating stakeholders.

Stage 4

Develop and deliver customised PQC training programs.

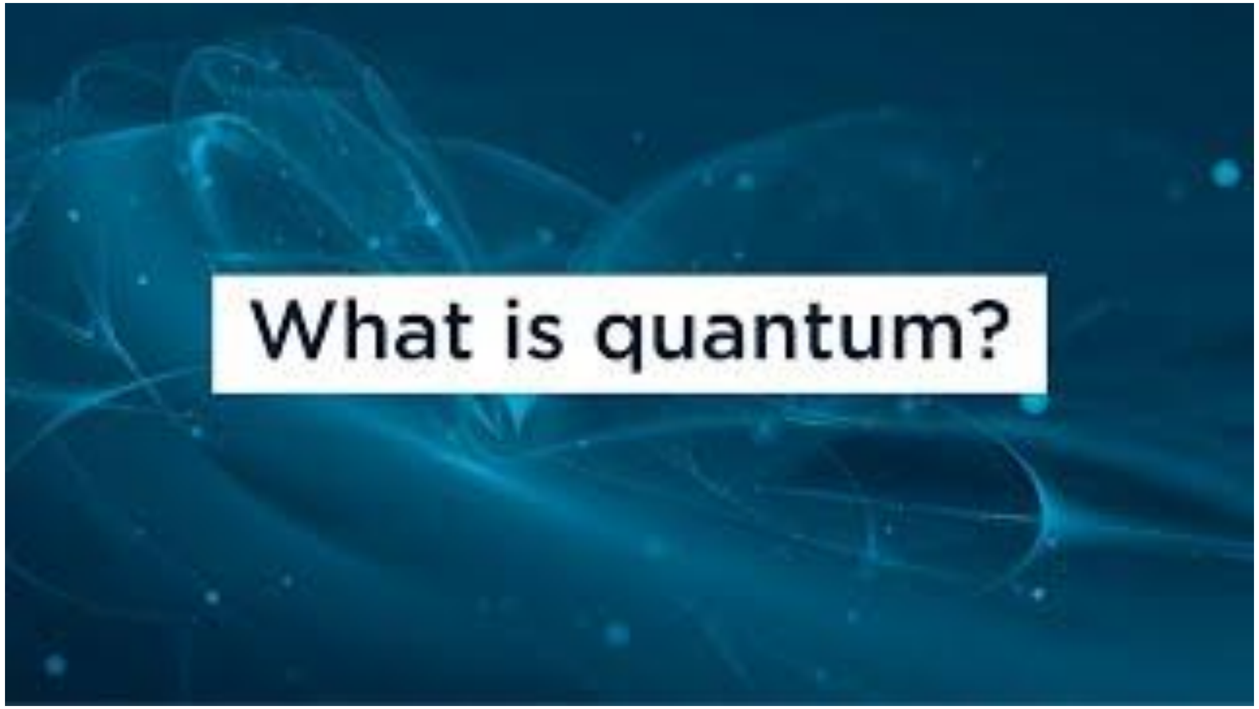
Stage 5

Formulate and evaluate participants transition implementation plans.

MONASH University

OCSC Oceania Cyber Security Centre

DEPARTMENT OF STATE UNITED STATES OF AMERICA



https://www.youtube.com/watch?v=17bvUUqCCzE

The **Quantum Technologies Future Science Platform** - help build world class quantum capability.

Collaboration with universities and industry, develop next generation solutions using quantum technologies.

Australia can continue playing a key role in the emerging global quantum industry.

The future is quantum and we are working to ensure Australia is ready for it.

- PQC, NIST PQC Process
- PQC Migration
- PQC and IoT

Resisting Quantum Attacks on Public-Key Crypto

- Two main countermeasure approaches investigated:
    - Quantum-Safe Cryptography
        - Aka Post-Quantum Cryptography (PQC)
        - Public key cryptosystems based on computational problems resistant even to quantum computer attacks
        - legitimate parties use only classical computers
            - “plug-in” replacement to quantum-insecure public key crypto.
        - Active research topic in cryptography for > 20 years

Resisting Quantum Attacks on Public-Key Crypto

- Two main countermeasure approaches investigated (cont):
  - Quantum Cryptography
    - Aka Quantum Key Distribution (QKD)
    - **Key exchange protocol** resistant even to quantum computer attacks
    - legitimate parties use quantum communication/ computation computers (not plug-in replacement for quantum-insecure public-key crypto, need special quantum hardware).
    - Requires quantum-safe classical authentication
    - Will not discuss further in this talk

NIST PQC

NIST (US) PQC standardization process: solicit, evaluate and standardise quantum-resistant public-key cryptosystems:

- Nov. 2017: PQC algorithm submissions deadline
  - 69 algorithms submitted (public key encryption and signatures)
  - Initiate 5 year analysis/evaluation phase
- Jan. 2019: Second round algorithms selected (26)
- Jul. 2020: Third round algorithms selected (7)
- Jul. 2022: New PQC standard algorithms selected (4)
- ~ 2022-23: New PQC standards developed
- **Goal:** ready for PQC deployment in by ~ 2024-27

NIST PQC Process: Current Status

Selected PQC standards:

- PKE/KEMs:
  - **Kyber** (Structured lattices)
- Signatures:
  - **Dilithium** (Structured Lattices)
  - **Falcon** (Structured lattices)
  - **SPHINCS+** (symmetric key / hash functions)

NIST PQC Process: Current Status

- Round 4 PKE/KEMs:
    - **BIKE** (Code)
    - **Classic McEliece** (Code)
    - **HQC** (Code)

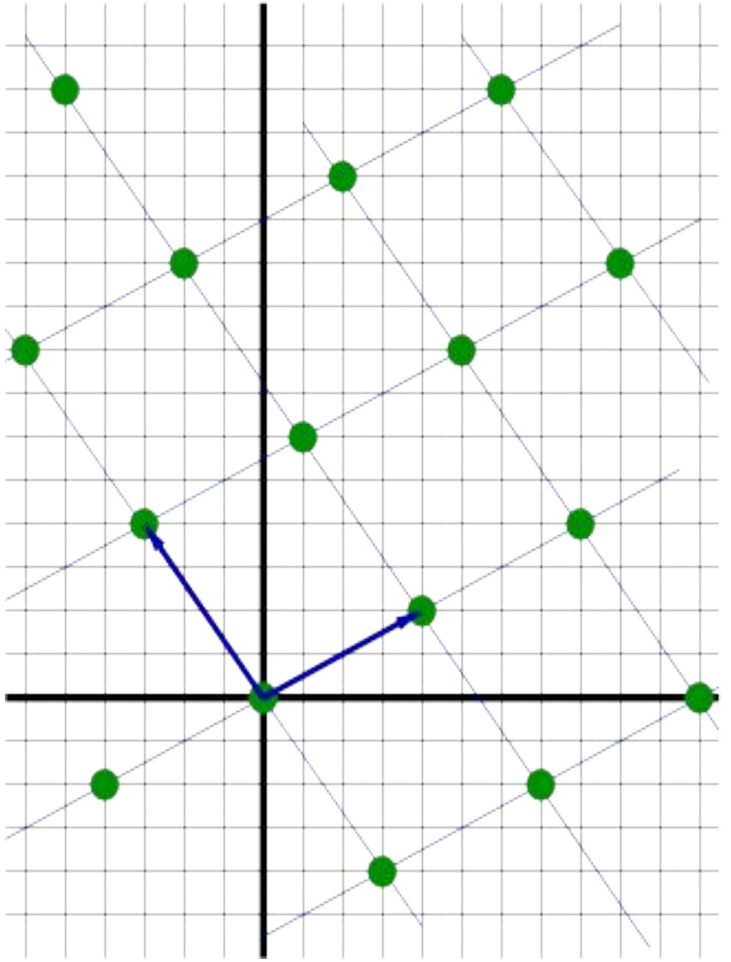
- Call for Additional Digital Signature Schemes:
    - NIST call for submission of additional efficient PQ signature proposals not based on structured lattices. Due 1 June 2023.
        - "backup" fast/short signature standard in case unexpected vulnerabilities discovered in lattice-based schemes
        - Possibly short/fast signatures for low-resource apps
    - Submission from CSIRO: eMLE-Sig 2.0

Quantum-Safe Crypto: Approaches – Lattices & Codes

- Linear Equations with errors – Codes & Lattices
  - Idea inspired by Error Correction Codes
  - Add ‘small’ errors to a linear equation to make it hard to solve: $y = A*x + e$
  - Encode a message x by an expanding linear transformation (add redundancy)
  - Can decode if noise e is sufficiently `small’
    - Easy to decode for special codes (wireless communication)
    - Computationally hard to decode for “random-looking” linear codes in high dimension
- Codes & Lattices: different ways to measure `small’

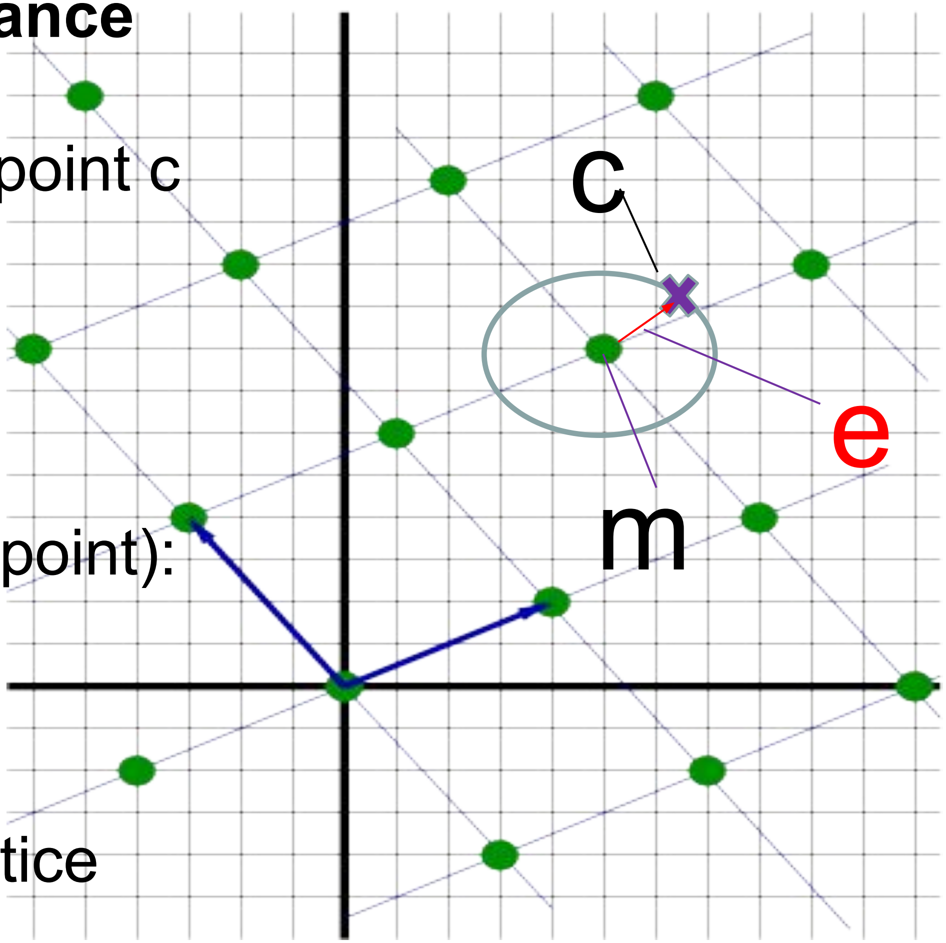
Lattice-based Cryptography: Idea (1)

- **Lattice** = periodic grid of points in space
- Generated by some set of **basis** vectors
  - E.g. (right) lattice in 2-D (green points = lattice, basis in blue)
- Can be easily defined mathematically in **any dimension n**
  - hard to visualise/draw for n > 3!
- **Fact:** **geometric** problems in lattices seem to be computationally infeasible (run time **exponential in n**) for large dimension n
  - Even against **quantum** computers!
- **Lattice-based crypto**: design pub-key encryption so breaking it requires solving a hard geometric lattice problem!



Lattice-Based Cryptography: Idea (2)

- **Hard geometric lattice Problem**: **Bounded-Distance Decoding** (**BDD**)
  - Given a basis B of a (high-dim.) lattice and a point c close to a lattice point m, compute m
- **Idea of Public-key encryption:**
  - **Pub key pk:** basis B for lattice
  - **Private key sk:** decoding **trapdoor** for lattice
  - **Encrypt(m):** to encrypt a message m (lattice point):
    - choose random short error vector e
    - Compute c = m + e
    - Ciphertext = c
  - **Decrypt(c, sk):** use sk to compute closest lattice point m to c.
- **Security**: hard to solve BDD without sk!



Quantum-Safe Crypto: Approaches – Lattices & Codes

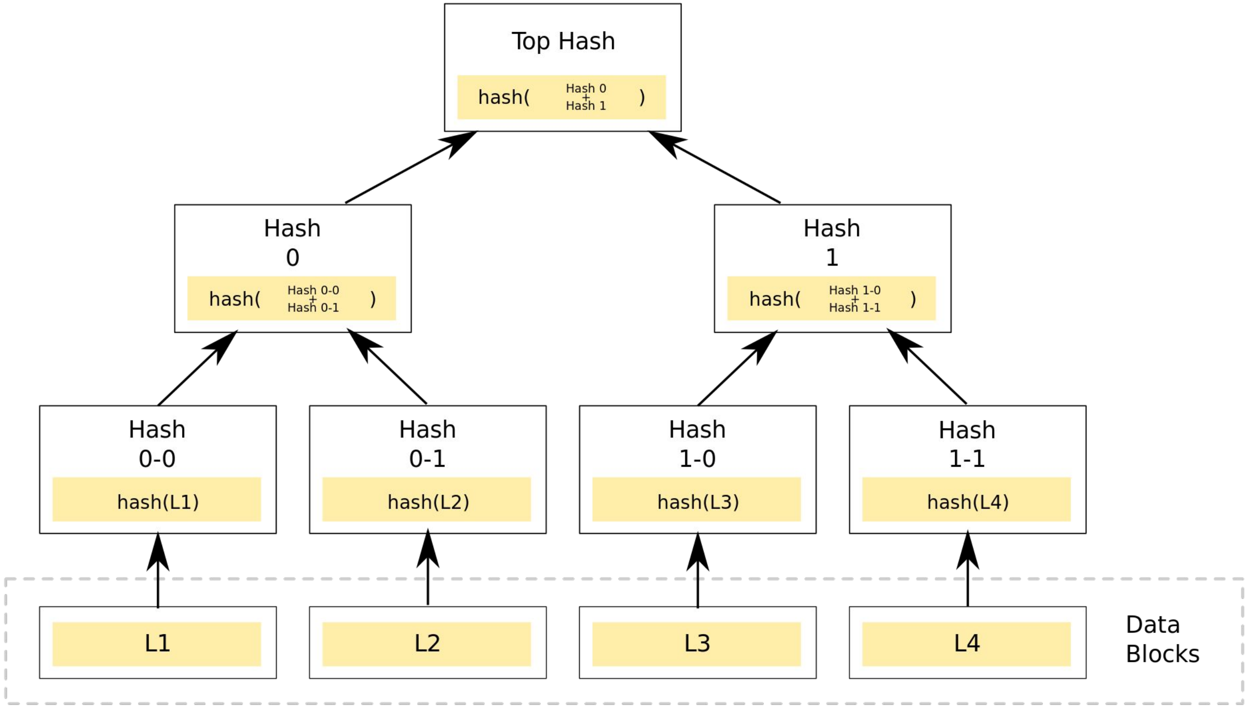
- Lattices: Performance and Security
  - Security:
    - best known attack time ~ $2^{O(k)}$ for key length k
    - But exponent constant is quite small → **moderately large key/ciphertext/signature lengths**
    - Studied in math & comp sci. since 1980s
  - Performance: With practical structured lattices (MLWE/RLWE/NTRU problems):
    - **fast algorithms (~ ECC or faster) and**
    - **moderately short keys/ctxts (~10x-40x ECC length)**

Quantum-Safe Crypto: Approaches – Lattices & Codes

- Codes: Performance and Security
  - Security:
    - best known attack time ~ $2^{O(k)}$ for key length k
    - But exponent constant is quite small → **moderately large key/ciphertext/signature lengths**
    - Studied in math & comp sci. since 1950s
    - McEliece PKC (1977) – based on Goppa codes
  - Performance: With original McEliece (one of the NIST Round 4 KEMs)
    - **Moderately fast algorithms**
    - **very short ctxts (~128 bytes)**
    - **Very long keys (> 100kB)**
    - **Structured codes can improve performance**
  - **Difficult to implement signatures!**

Quantum-Safe Crypto: Approaches – Symmetric-key approaches (digital signatures only)

- Idea:
  - Use well established symmetric-key algorithms
    - Public key = Merkle tree hash of many one-time signature keys (**short**)
    - Signature = one-time signature (reveal sk) + Merkle auth path (siblings of all nodes on the path from leaf to root)
      – long signature!
- Security: well understood
- Performance:
    - short public key
    - Long signature and slow algorithms



Challenges in PQC migration

- **Performance** characteristics are very different compared to classical PKC.
    - **Slower speed**
        - Can be improved with better SW/HW implementations in the future
    - **Larger sizes** (public key, ciphertext/signature)
        - No easy way to improve without redesigning the scheme

Challenges in PQC migration

- **Challenge:** Many existing applications/protocols were designed based on the performance characteristics of classical PKC.
    - E.g. Key/ciphertext may not fit into one packet in network communication.
        - Fragmentation issues for UDP-based protocols e.g. IKE
    - Require dedicated solutions for PQC migration!
- **Challenge:** Lack of confidence in PQC.
    - Classical PKC has been analyzed and used in practice for decades.
        - Backward compatibility issues
    - Potential new attacks against PQC.
    - Require **hybrid** solutions.

PQC speed comparison (128-bit security, scaled to ms on 1GHz Intel CPU with AVX2 and AES-NI)

Scheme	**KeyGen**	**Encap/Sign**	**Decap/Verify**
Kyber512 (AES PRG)	0.02188	0.028592	0.02098
Dilithium2 (AES PRG)	0.070548	0.194892	0.072633
Falcon512	19.872	0.386678	0.08234
SPHINCS+-Haraka-128f-simple	0.482332	12.196792	0.799808
SPHINCS+-Haraka-128s-simple	30.075604	240.763926	0.308774

PQC size comparison (128-bit security, in bytes)

Scheme	**Public Key**	**Ciphertext/Signature**
Kyber512	800	768
Dilithium2	1312	2420
Falcon512	897	666
SPHINCS+-128f	32	17088
SPHINCS+-128s	32	7856
ECDHE (secp256r1)	32	32
RSA-2048 Signature	256	256

Note: The Maximum Transmission Unit (MTU) of the Ethernet is ~1500 bytes.

5W1H in PQC migration

- Who, What, When, Where, Why, How
  - **Who:** Almost **every** organisation
  - **When:** Need to start **now**
  - **Why:** Quantum attacks; Fast emerging quantum technology
- This Section will focus on **What**, **Where**, and **How**:
  - What (libraries etc.) can be used for PQC migration?
  - Where changes need to be made in certain protocols/applications?
  - How to make changes? (i.e. the migration strategy)

Common Internet communication protocols

- TLS, IPSec, SSH, …
- Goals:
  - Want to have (virtual) **secure channel** between two points
  - **Secure** in terms of
    - Confidentiality
    - Integrity
    - Authentication
- Use **Public Key Cryptography (PKC)** to authenticate the peers and establish a shared (symmetric) secret key; then use symmetric key cryptography for bulk data
  - **Key agreement protocol** for establishing the shared secret
  - **Digital signature** for authentication (certificate)

Architectural design of common libraries

- Cryptography module + Protocol module
- **Cryptography** module:
  - Implement ciphers (PKC and symmetric), e.g. RSA, AES, SHA-256, …
- **Protocol** module:
  - Implement the actual protocol e.g. TLS, IKE, …
  - Depend on the cryptography module for ciphers
- Examples:
  - TLS library OpenSSL: libcrypto (cryptography module) + libssl (protocol module)
  - IPSec application StrongSwan: libstrongswan (cryptography module) + libcharon (protocol module)

PQC migration: Pure PQC

- Replace classical key agreement protocol and/or digital signatures with PQC analogue
    - E.g. ECDHE → Kyber; RSA signature → Dilithium
- Changes in cryptography:
    - Implement PQC algorithms
- Changes in protocol:
    - Support PQC ciphers (e.g. add to the cipher suite/DH group number)
    - Need to support Key Encapsulation Mechanism (KEM) in key agreement (see TLS migration for an example)
    - May need dedicated solutions for PQC performance characteristics (e.g. IKE_INTERMEDIATE [RFC9242] in IKEv2)

Limitations of Pure PQC

- Backward **incompatible** with classical PKC
- Lack of confidence in PQC
  - Theories have been studied for years, **instantiations are not**.
    - Potential new attacks against certain PQC constructions/schemes/implementations.
  - Risk management.
    - "***Diversification*** *is the only free lunch*" – Harry Markowitz, Nobel Prize laureate.
- Need **hybrid** between classical PKC and PQC
  - Security depends on the **strongest** of the two.
  - We focus on **key agreement**, as hybrid signature/certificate isn't well understood.

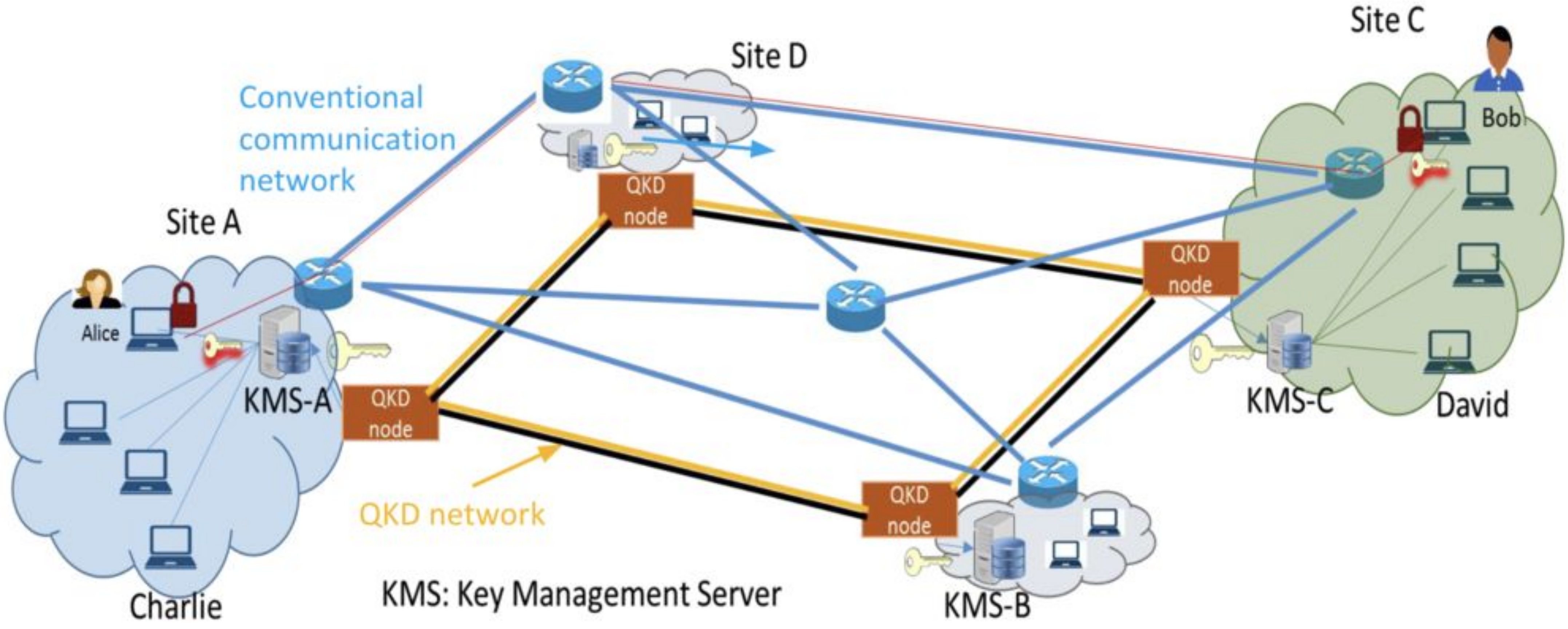
PQC migration: Hybrid between classical PKC and PQC

- **Non-composite** hybrid:
    - Modify the design and state machine of the protocol.
        - Perform both classical and PQC key agreements, then combine the generated secret values.
    - Keep the changes in cryptography minimal.
        - Leave the ciphers (ECDHE, Kyber, etc.) “as it is”.
    - Example: Multiple Key Exchanges in IKEv2 [RFC9370]

PQC migration: Hybrid between classical PKC and PQC

- **Composite** hybrid:
  - Define “hybrid” ciphers in cryptography.
    - Perform both classical and PQC algorithms. The generated secret value contains entropy from both.
    - E.g. x25519_kyber512
  - Keep the protocol “as it is”.
  - Example: Hybrid key exchange in TLS 1.3 (IETF draft)

PQC migration: Hybrid between PQC and QKD



Basquana: https://www.qkdnetworkcanadauk.com/

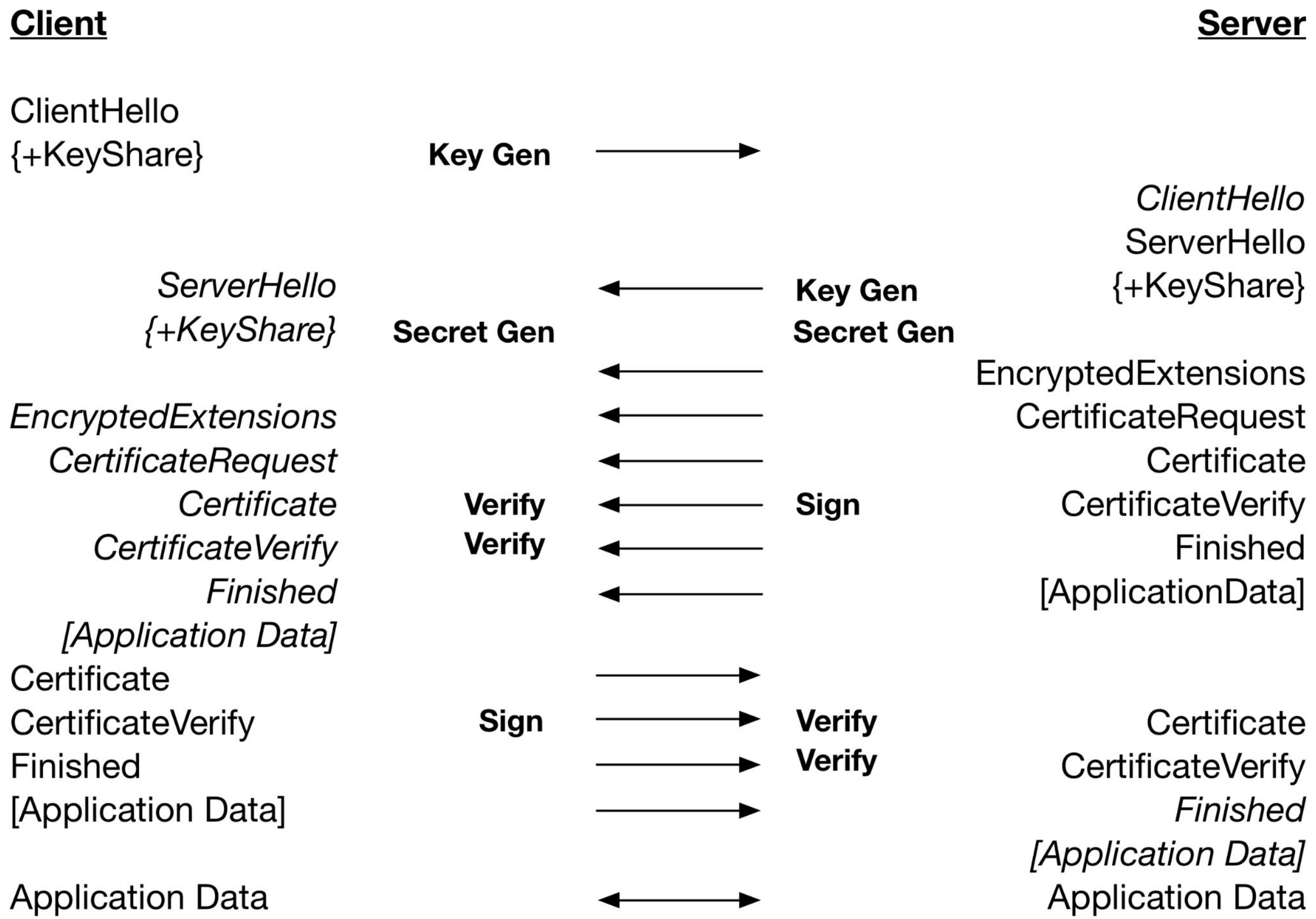
Cryptographic libraries implementing PQC

- Open Quantum Safe (liboqs):
  - https://openquantumsafe.org/
  - C library
  - Developed by University of Waterloo, Canada
  - Include reference implementations of NIST PQC selected algorithms for standardization, Round 4 KEMs, FrodoKEM, NTRUPrime
  - Provide common APIs for these algorithms
  - Preliminary integrations in TLS and SSH
    - https://openquantumsafe.org/applications/tls.html
    - https://openquantumsafe.org/applications/ssh.html

Cryptographic libraries implementing PQC

- Bouncy Castle
  - https://www.bouncycastle.org/
  - Java/C# library
  - Developed by Australian charity Legion of the Bouncy Castle Inc.
  - Include implementations of NIST PQC selected algorithms for standardization, Round 4 KEMs, FrodoKEM, NTRUPrime, NTRU, SABER, Picnic, XMSS, Leighton-Micali.
  - Monash University contributed to the initial implementations of NTRU, Falcon, Kyber, NTRUPrime, Dilithium.

TLS 1.3 handshake



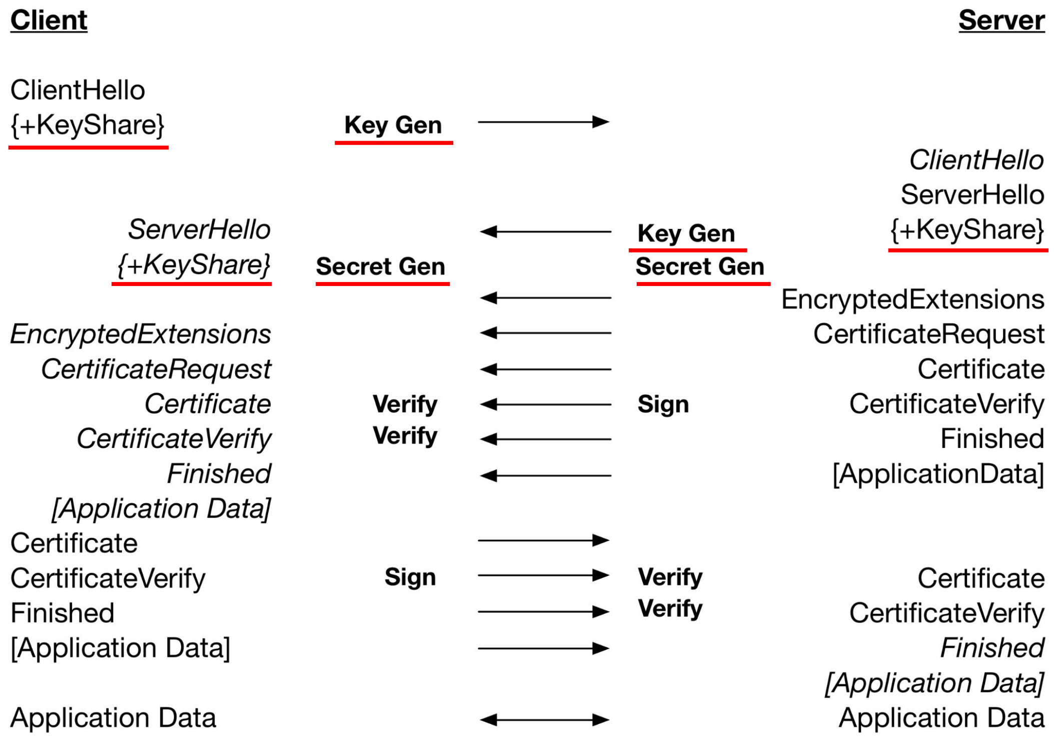
Client | **Server**

ClientHello
{+KeyShare} **Key Gen** →

ClientHello
ServerHello
ServerHello ← **Key Gen** {+KeyShare}
{+KeyShare} **Secret Gen** **Secret Gen**
← EncryptedExtensions
EncryptedExtensions ← CertificateRequest
CertificateRequest ← Certificate
Certificate **Verify** ← **Sign** CertificateVerify
CertificateVerify **Verify** ← Finished
Finished ← [ApplicationData]
[Application Data]
Certificate →
CertificateVerify **Sign** → **Verify** Certificate
Finished → **Verify** CertificateVerify
[Application Data] → *Finished*
[Application Data]
Application Data ↔ Application Data

Supporting KEM in TLS 1.3

- Transformation of the **Key Exchange** scheme into **Key Encapsulation Mechanisms (KEM)** scheme
- Adopted the proposition of CRYSTALS-Kyber [17]



Client **Server**

ClientHello {+KeyShare} Key Gen →

ClientHello

ServerHello {+KeyShare}

ServerHello *{+KeyShare}* Secret Gen ← Key Gen Secret Gen

← EncryptedExtensions

EncryptedExtensions ← CertificateRequest

CertificateRequest ← Certificate

Certificate Verify ← Sign CertificateVerify

CertificateVerify Verify ← Finished

Finished ← [ApplicationData]

[Application Data]

Certificate →

CertificateVerify Sign → Verify Certificate

Finished → Verify CertificateVerify

[Application Data] → *Finished*

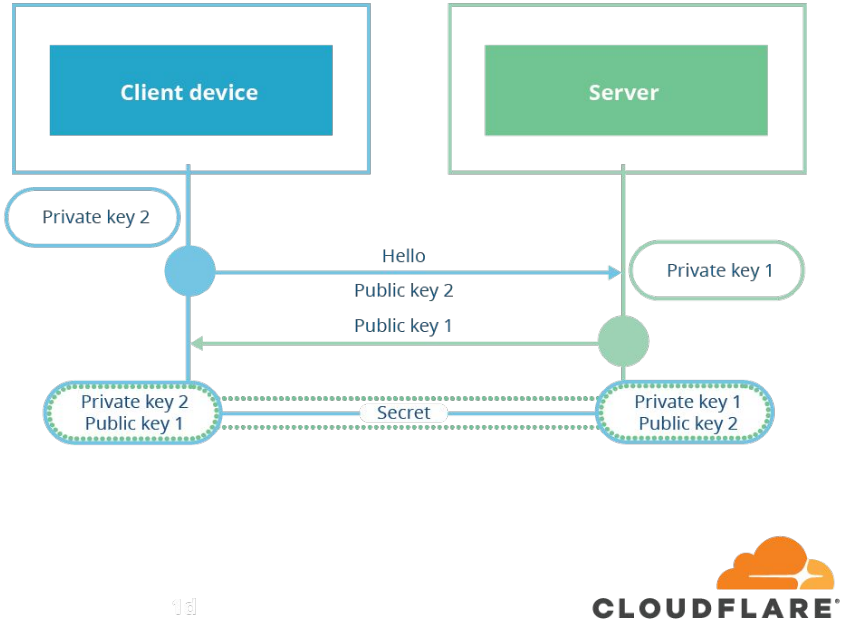
[Application Data]

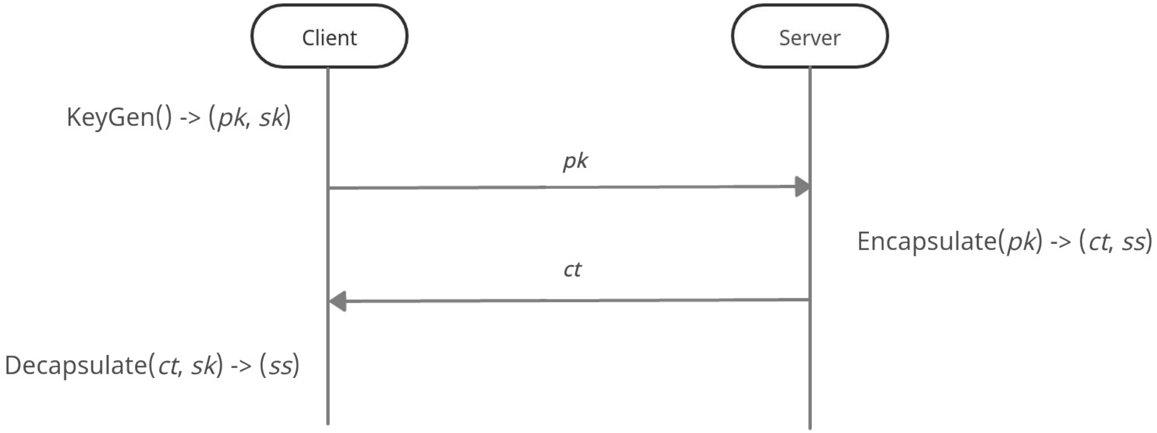
Application Data ↔ Application Data

17. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: ***CRYSTALS - kyber: A cca-secure module lattice-based KEM***. In: EuroS&P. pp. 353–367. IEEE (2018)

Supporting KEM in TLS 1.3

DH 1.3 handshake



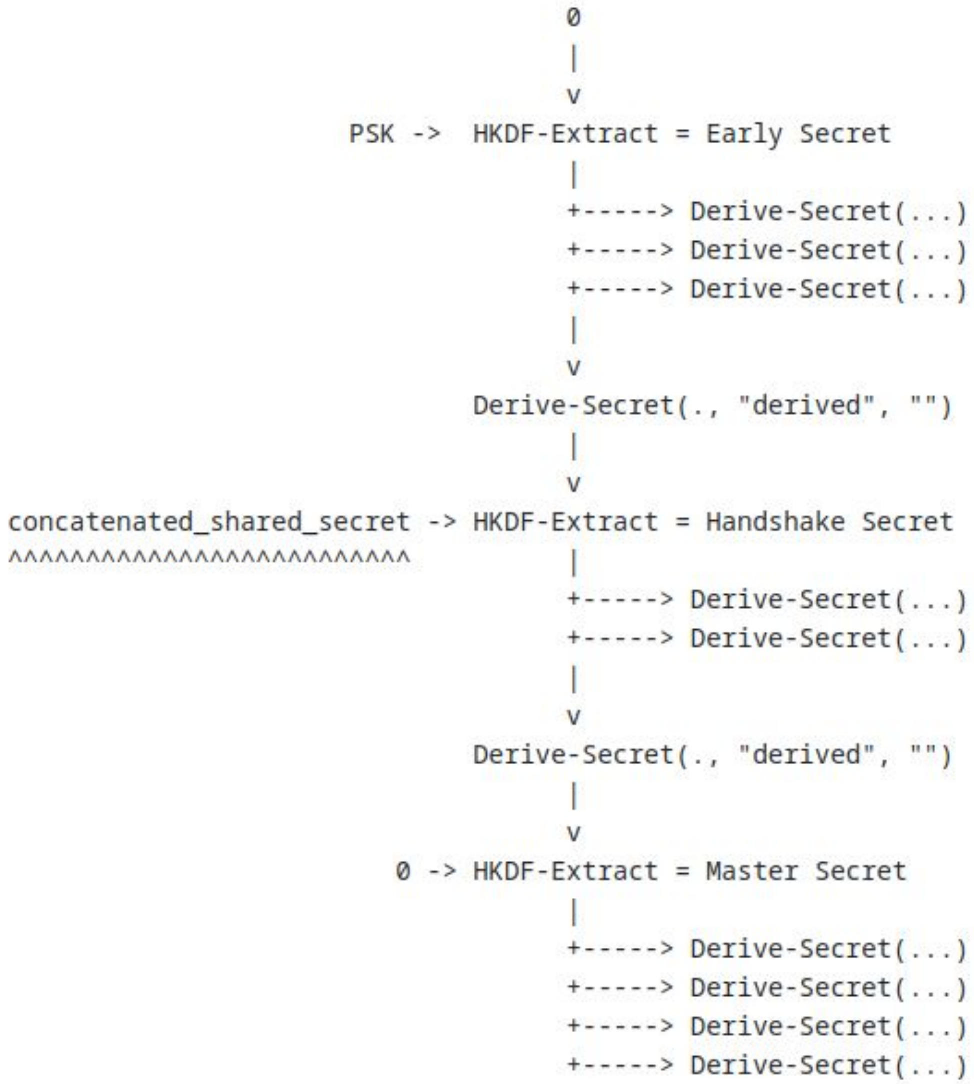

Client
Server
KeyGen() -> (*pk*, *sk*)
pk
Encapsulate(*pk*) -> (*ct*, *ss*)
ct
Decapsulate(*ct*, *sk*) -> (*ss*)

Diffie-Hellman Key Exchange scheme -> KEM scheme

- Public Key 2 -> pk
- Public Key 1 -> ct

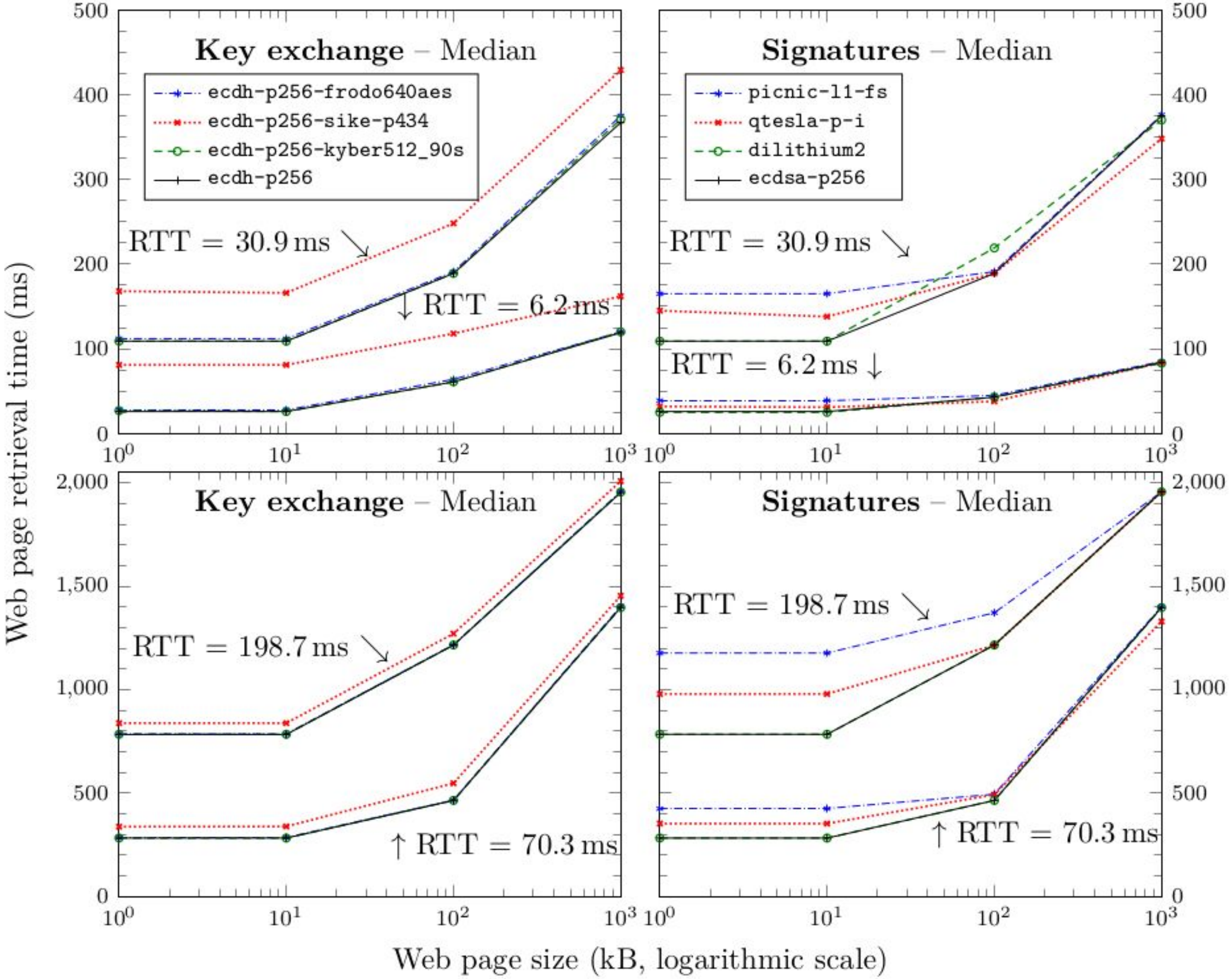
Hybrid key exchange in TLS 1.3

- IETF draft by Stebila et al. https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/
- Composite hybrid using simple concatenation approach:
    - Concatenation of public values in key exchange
    - Concatenation of shared secrets
        - Drop-in replacement of the (EC)DHE shared secret in key schedule



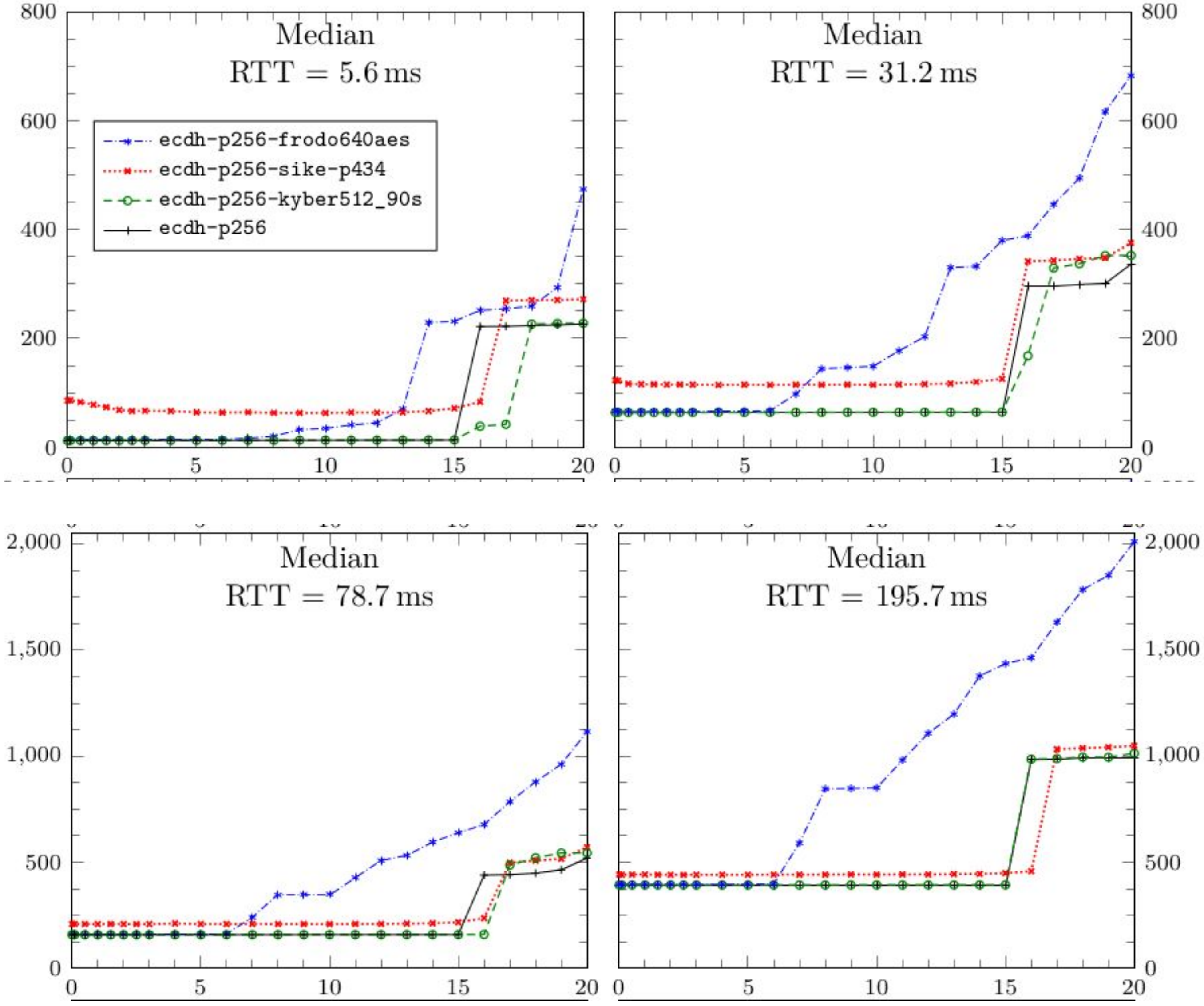
```
                                    0
                                    |
                                    v
                      PSK ->  HKDF-Extract = Early Secret
                                    |
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
                                    |
                                    v
                              Derive-Secret(., "derived", "")
                                    |
                                    v
concatenated_shared_secret -> HKDF-Extract = Handshake Secret
^^^^^^^^^^^^^^^^^^^^^^^^^^          |
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
                                    |
                                    v
                              Derive-Secret(., "derived", "")
                                    |
                                    v
                         0 -> HKDF-Extract = Master Secret
                                    |
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
                                    +-----> Derive-Secret(...)
```

Performance of PQ TLS 1.3 (real network)

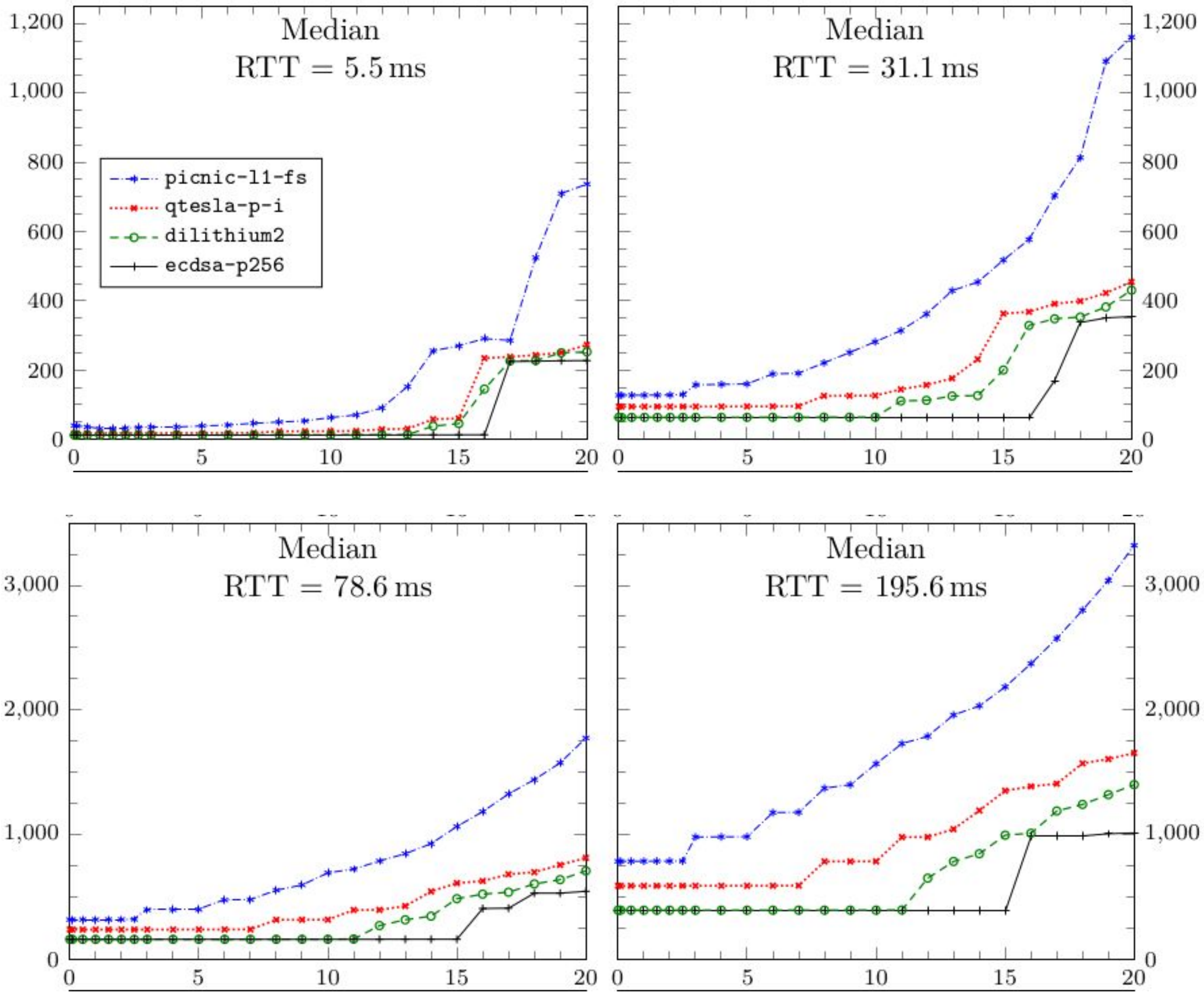


- Paquin et al. (2020) measured the performance of PQ TLS 1.3 with liboqs.
    - Hybrid key exchange
    - PQC signatures
- In real network environment:
    - Hybrid key exchange with Kyber/FrodoKEM only adds very small overhead
    - PQC signatures may add bigger overhead
- In unreliable network environment (high packet loss rate), performance degradation is more significant with PQC due to larger sizes (see next slides).

Performance of PQ TLS 1.3 (simulated unreliable network)

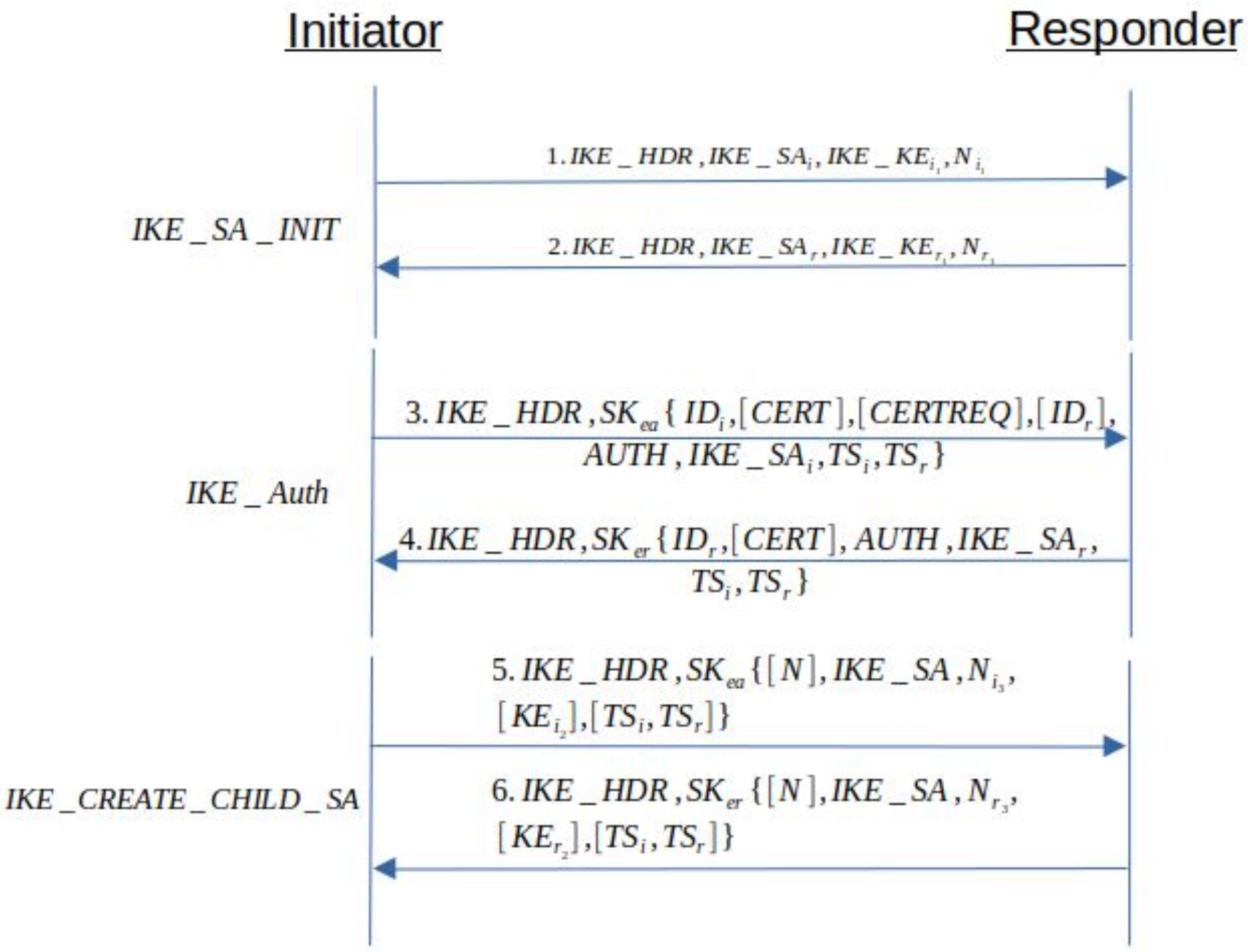


Performance of PQ TLS 1.3 (simulated unreliable network)



Internet Key Exchange (IKE) v2

- **UDP**-based protocol.
  - TCP mode isn't widely deployed.
- Set up Security Association (SA) in IPSec.
  - Key exchanges
    - IKE_SA_INIT for creating IKE SA
    - IKE_CREATE_CHILD_SA for creating additional child SA
  - Authentication (IKE_AUTH)
- We focus on key exchange.


Initiator
Responder
IKE_SA_INIT
$1. IKE\_HDR, IKE\_SA_i, IKE\_KE_{i_1}, N_{i_1}$
$2. IKE\_HDR, IKE\_SA_r, IKE\_KE_{r_1}, N_{r_1}$
IKE_Auth
$3. IKE\_HDR, SK_{ea}\{ID_i, [CERT], [CERTREQ], [ID_r], AUTH, IKE\_SA_i, TS_i, TS_r\}$
$4. IKE\_HDR, SK_{er}\{ID_r, [CERT], AUTH, IKE\_SA_r, TS_i, TS_r\}$
$IKE\_CREATE\_CHILD\_SA$
$5. IKE\_HDR, SK_{ea}\{[N], IKE\_SA, N_{i_3}, [KE_{i_2}], [TS_i, TS_r]\}$
$6. IKE\_HDR, SK_{er}\{[N], IKE\_SA, N_{r_3}, [KE_{r_2}], [TS_i, TS_r]\}$

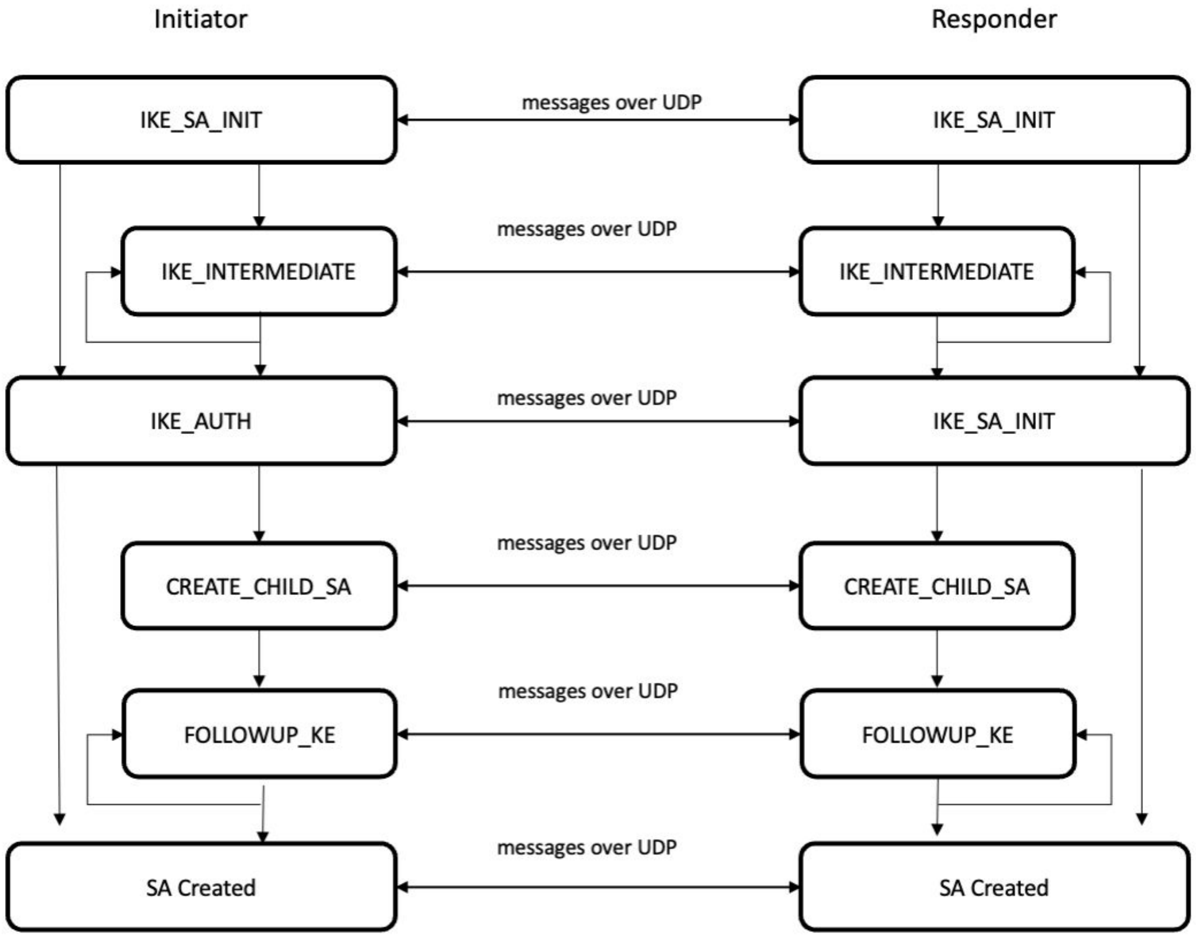
PQC migration in IKEv2

- PQC migration in StrongSwan 6.0 beta:
  - https://github.com/strongswan/strongswan/tree/six-beta
  - Implement PQC algorithms via oqs plugin in libstrongswan (cryptography module)
    - Depend on liboqs
  - Implement PQC key exchanges in libcharon (protocol module)
    - Pure PQC:
      - Direct substitution, using private DH group numbers
      - Not using IKE_INTERMEDIATE (next slide)
        - Potential IP fragmentation problems
    - (Non-composite hybrid) Multiple Key Exchanges [RFC9370]
      - Example configuration: https://github.com/strongswan/strongswan/tree/six-beta/testing/tests/ikev2/rw-cert-qske

IKE_INTERMEDIATE [RFC9242]

- IKEv2 is **UDP**-based:
    - Unlike TCP-based protocol, UDP doesn't have segmentation mechanism.
    - Cause **IP fragmentation** (very undesirable) if key exchange messages can't fit into one packet.
        - **Problems:** performance, firewall, potential DoS attack, …
- IKEv2 has a message fragmentation mechanism [RFC7383]
    - However, IKE_SA_INIT key exchange messages can't be fragmented.
- IKE_INTERMEDIATE:
    - Do PQC key exchange after initial IKE_SA_INIT (before IKE_AUTH).
    - IKE_INTERMEDIATE messages are fragmented by IKEv2 fragmentation.
    - Example: IKEv2 multiple key exchanges [RFC9370]
        - IKE_SA_INIT for classical key exchange
        - IKE_INTERMEDIATE for PQC key exchange

Multiple Key Exchanges in IKEv2 [RFC9370]



Future Research

- Problems with existing PQC migration techniques:
  - Extra **complexity**: entangled state machine, overheads, …
  - Require **ad-hoc** solution for every application/protocol:
    - E.g. > 8,000 lines of code changes in StrongSwan 6.0 beta to implement IKEv2 multiple key exchanges.
- **Question:** Could we have better **architectural** design for PQC migration?
  - Simplicity
  - Modularity

Future Research

- Understand the **scope** of PQC migration.
    - Beyond common protocols, there exist **custom** protocols/applications in organizations.
    - White House asked US government departments and agencies to submit cryptographic system inventory by May 4, 2023 (then reevaluate annually).
        - Submit funding assessments 30 days after submission of cryptographic system inventory.
- **Question:** How to figure out the **exact scope** of PQC migration for a particular organization?
- CSIRO is engaging with industrial partners for both questions!

Future Research

- 4 types of **crypto agility** in the context of PQC (Alnahawi et al., 2023):
  - Algorithm and Protocol Agility
    - E.g. Hybrid key exchange
  - API Agility
    - E.g. Common APIs in liboqs, Bouncy Castle
  - **Design Agility**
    - How to design PQC crypto-agile protocols/applications in the future?
  - **Hardware Agility**
    - How to design PQC crypto-agile hardwares (FPGA, crypto coprocessor etc.)?
    - Could we reuse existing crypto (e.g. RSA) coprocessors for PQC?

Unique challenges of PQC migration on IoT

- Resource constraints:
  - Computational Power
  - **Memory** (stack, code)
  - **Energy** (e.g. battery powered)
- **Difficult** to patch/update:
  - Require **hardware agility**!
- **Hostile** physical environment:
  - Side-channel attacks
  - Require **side-channel resistant** implementation!
- We mainly focus on ARM Cortex-M4 (NIST PQC evaluation platform for embedded devices)

Pqm4 cryptography library

- Collection of PQC implementations targeting ARM Cortex-M4
  - https://github.com/mupq/pqm4
  - Developed by Radboud University, Netherland
  - Currently version includes **assembly** optimized implementations for Kyber, BIKE, Dilithium, and Falcon; and reference implementation for HQC, SPHINCS+.
    - Some NIST Round 3 algorithms are available in older version.

PQC speed comparison (CPU cycles, 128-bit security, pqm4 implementation, assembly optimized if available)

Scheme	KeyGen	Encap/Sign	Decap/Verify
Kyber512 (AES PRG, speed opt.)	369,011	421,685	420,333
Kyber512 (AES PRG, stack opt.)	369,736	424,339	423,234
Dilithium2	1,597,999	4,111,596	1,571,804
Falcon512	179,772,454	17,649,735	480,619
SPHINCS+-sha256-128f-simple	15,388,375	382,533,954	21,150,671
SPHINCS+-sha256-128s-simple	985,367,046	7,495,603,716	7,165,875

PQC stack usage comparison (bytes, 128-bit security, pqm4 implementation, assembly optimized if available)

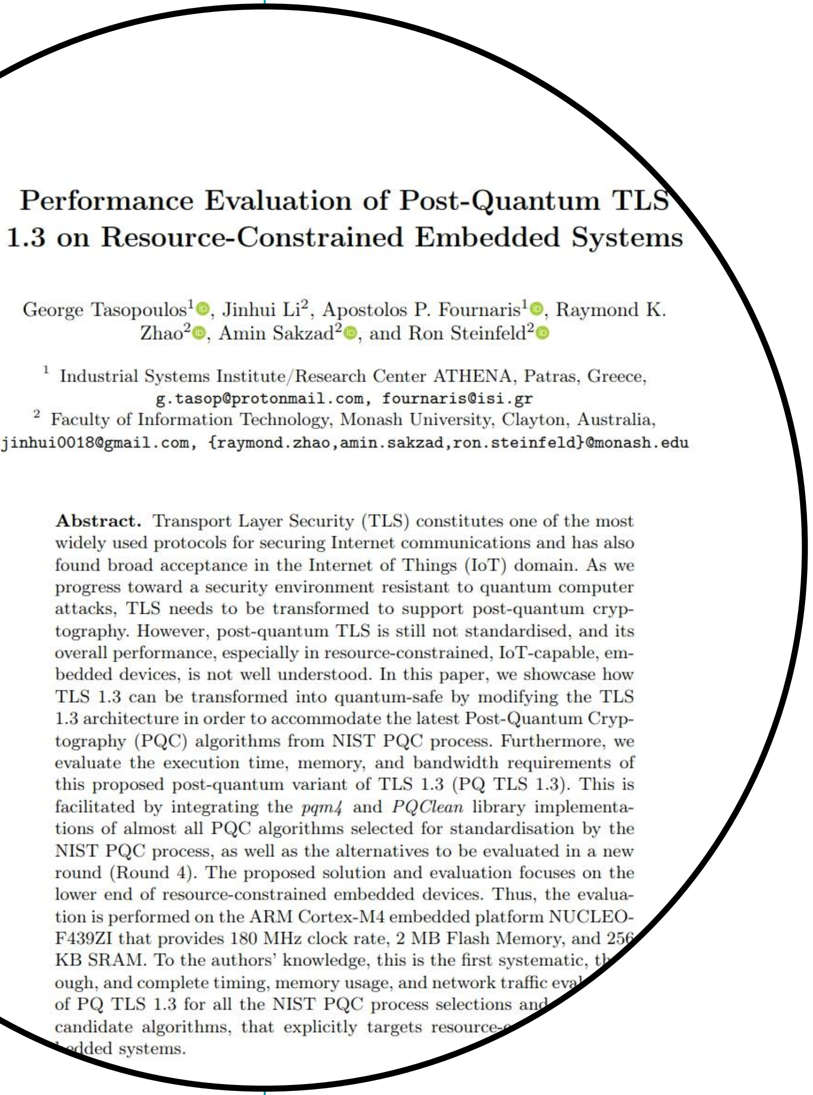
Scheme	KeyGen	Encap/Sign	Decap/Verify
Kyber512 (AES PRG, speed opt.)	5,076	6,180	6,188
Kyber512 (AES PRG, stack opt.)	3,012	3,100	3,116
Dilithium2	38,408	49,380	36,212
Falcon512	1,408	2,796	412
SPHINCS+-sha256-128f-simple	2,124	2,188	2,676
SPHINCS+-sha256-128s-simple	2,340	2,408	1,980

PQC code size comparison (bytes, 128-bit security, pqm4 implementation, assembly optimized if available)

Scheme	.text	.bss	Total
Kyber512 (AES PRG, speed opt.)	15,784	0	15,784
Kyber512 (AES PRG, stack opt.)	13,052	0	13,052
Dilithium2	18,480	0	18,480
Falcon512	82,821	27,648	110,469
SPHINCS+-sha256-128f-simple	4,504	0	4,504
SPHINCS+-sha256-128s-simple	4,776	0	4,776

.data size is 0 for all schemes in the Table.

Our work



Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems

George Tasopoulos[1], Jinhui Li[2], Apostolos P. Fournaris[1], Raymond K. Zhao[2], Amin Sakzad[2], and Ron Steinfeld[2]

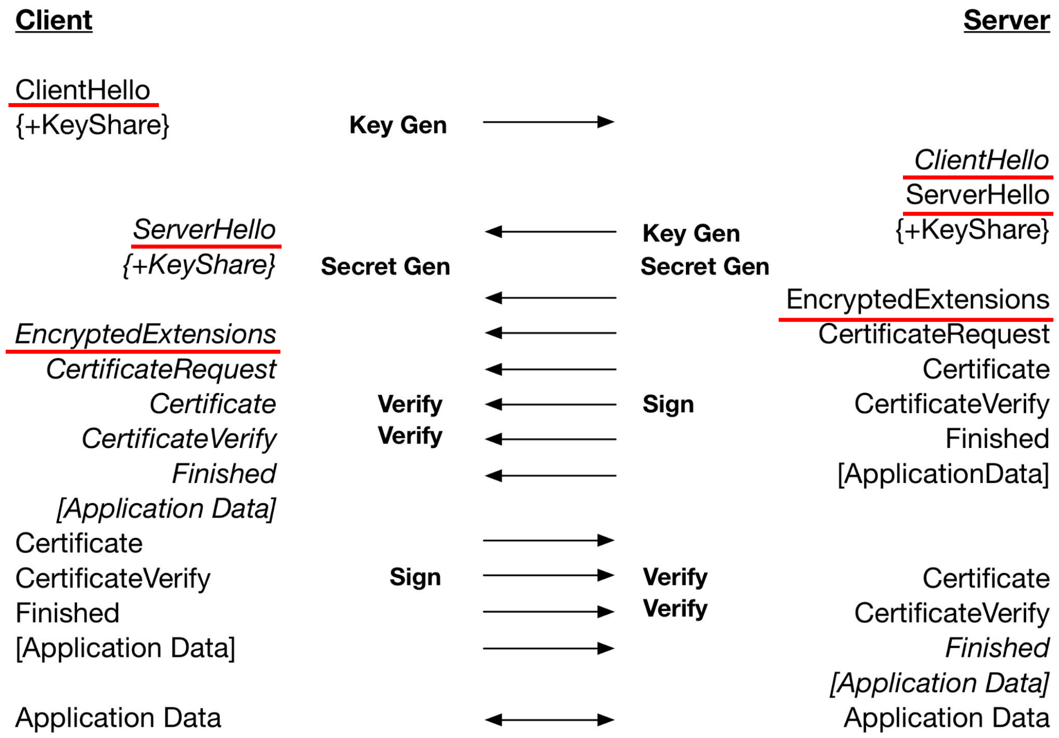
[1] Industrial Systems Institute/Research Center ATHENA, Patras, Greece,
g.tasop@protonmail.com, fournaris@isi.gr
[2] Faculty of Information Technology, Monash University, Clayton, Australia,
jinhui0018@gmail.com, {raymond.zhao,amin.sakzad,ron.steinfeld}@monash.edu

Abstract. Transport Layer Security (TLS) constitutes one of the most widely used protocols for securing Internet communications and has also found broad acceptance in the Internet of Things (IoT) domain. As we progress toward a security environment resistant to quantum computer attacks, TLS needs to be transformed to support post-quantum cryptography. However, post-quantum TLS is still not standardised, and its overall performance, especially in resource-constrained, IoT-capable, embedded devices, is not well understood. In this paper, we showcase how TLS 1.3 can be transformed into quantum-safe by modifying the TLS 1.3 architecture in order to accommodate the latest Post-Quantum Cryptography (PQC) algorithms from NIST PQC process. Furthermore, we evaluate the execution time, memory, and bandwidth requirements of this proposed post-quantum variant of TLS 1.3 (PQ TLS 1.3). This is facilitated by integrating the *pqm4* and *PQClean* library implementations of almost all PQC algorithms selected for standardisation by the NIST PQC process, as well as the alternatives to be evaluated in a new round (Round 4). The proposed solution and evaluation focuses on the lower end of resource-constrained embedded devices. Thus, the evaluation is performed on the ARM Cortex-M4 embedded platform NUCLEO-F439ZI that provides 180 MHz clock rate, 2 MB Flash Memory, and 256 KB SRAM. To the authors' knowledge, this is the first systematic, th ough, and complete timing, memory usage, and network traffic eval of PQ TLS 1.3 for all the NIST PQC process selections and candidate algorithms, that explicitly targets resource- edded systems.

- 12 NIST PQC algorithms from ***pqm4*** library
- Integration in ***WolfSSL's*** impl. of TLS 1.3
- Benchmarked them on a board with ***Cortex-M4***
- Performance evaluation of:
  - execution speed
  - memory requirements
  - communication sizes
- Complete overview of PQ TLS integration in resource-constrained embedded systems

Architectural Changes (1st change)

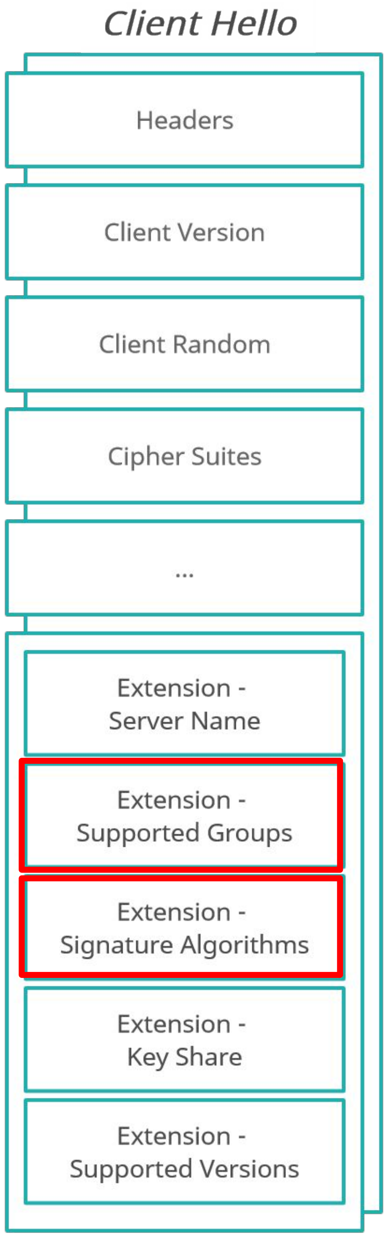
- Change in “ClientHello” and “ServerHello”
- First messages; protocol parameters
- Both have the *Extensions* field



Architectural Changes (1st change)

- Inside *Extensions* field:
  - *Extension - Supported Groups*
  - *Extension - Signature Algorithms*
- Codepoints
  - New codepoints for PQ algorithms
  - Both chosen according to Open Quantum Safe project
  - Benefit of interoperability of the projects

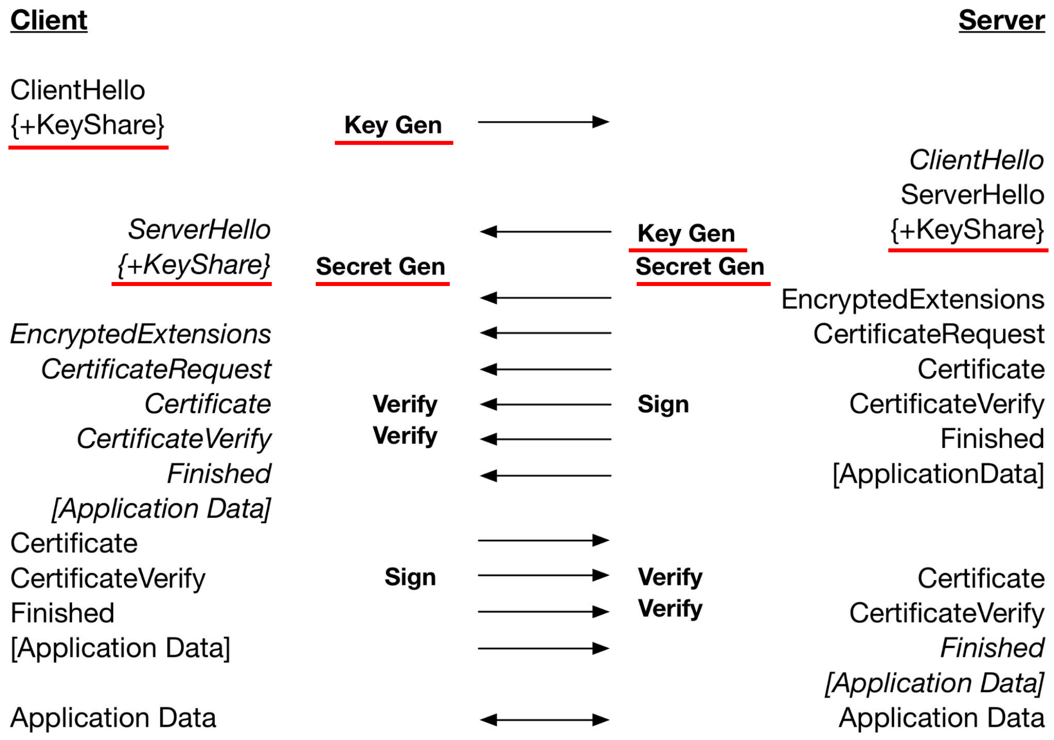
Scheme	Codepoints
ECDH SECP256R1	0x0017
FFDHE	0x0100
Kyber512	0x2F00
Kyber768	0x2F01
Lightsaber	0x2F03
Saber	0x2F04
…	…
RSA	0x0285
ECDSA	0x0206
Dilithium2	0x00D3
Dilithium3	0x00D5
…	…



Architectural Changes (2nd change)

- Transformation of the **Key Exchange** scheme into **Key Encapsulation Mechanisms (KEM)** scheme

- Adopted the proposition of CRYSTALS-Kyber [17]



Client

Server

ClientHello
{+KeyShare} **Key Gen** →

ClientHello
ServerHello
{+KeyShare}

ServerHello
{+KeyShare} **Secret Gen** ← **Key Gen** **Secret Gen**

← EncryptedExtensions

EncryptedExtensions ← CertificateRequest

CertificateRequest ← Certificate

Certificate **Verify** ← **Sign** CertificateVerify

CertificateVerify **Verify** ← Finished

Finished ← [ApplicationData]

[Application Data]

Certificate →

CertificateVerify **Sign** → **Verify** Certificate

Finished → **Verify** CertificateVerify

[Application Data] → *Finished*

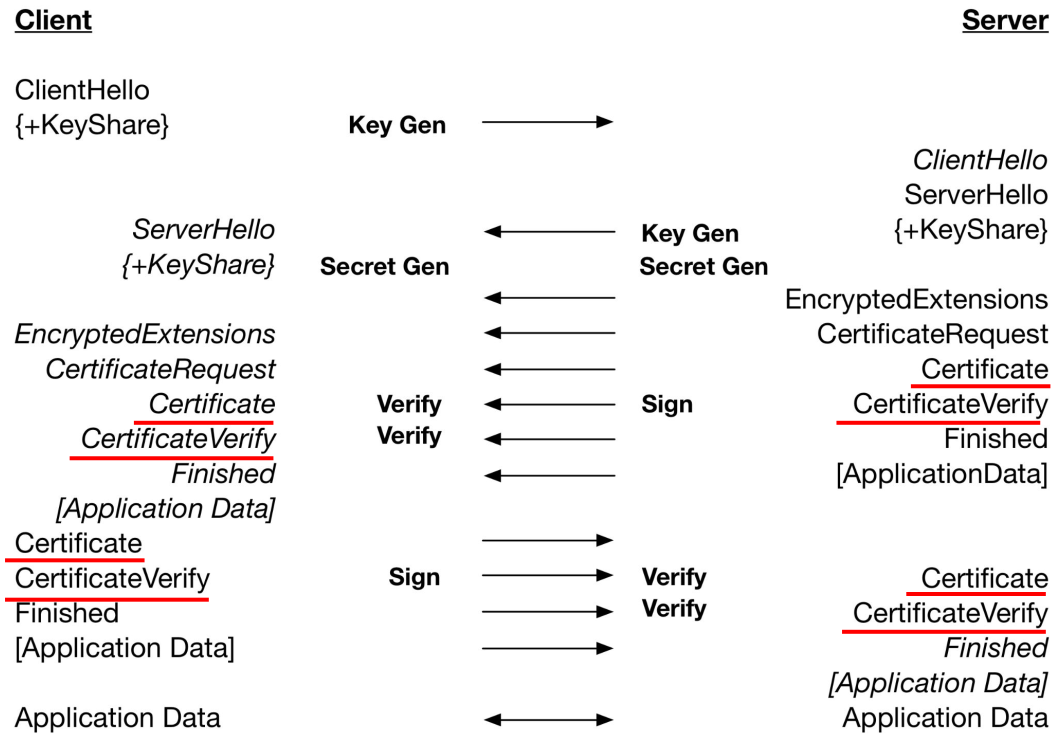
[Application Data]

Application Data ↔ Application Data

[17] Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: ***CRYSTALS - kyber: A cca-secure module lattice-based KEM***. In: EuroS&P. pp. 353–367. IEEE (2018)

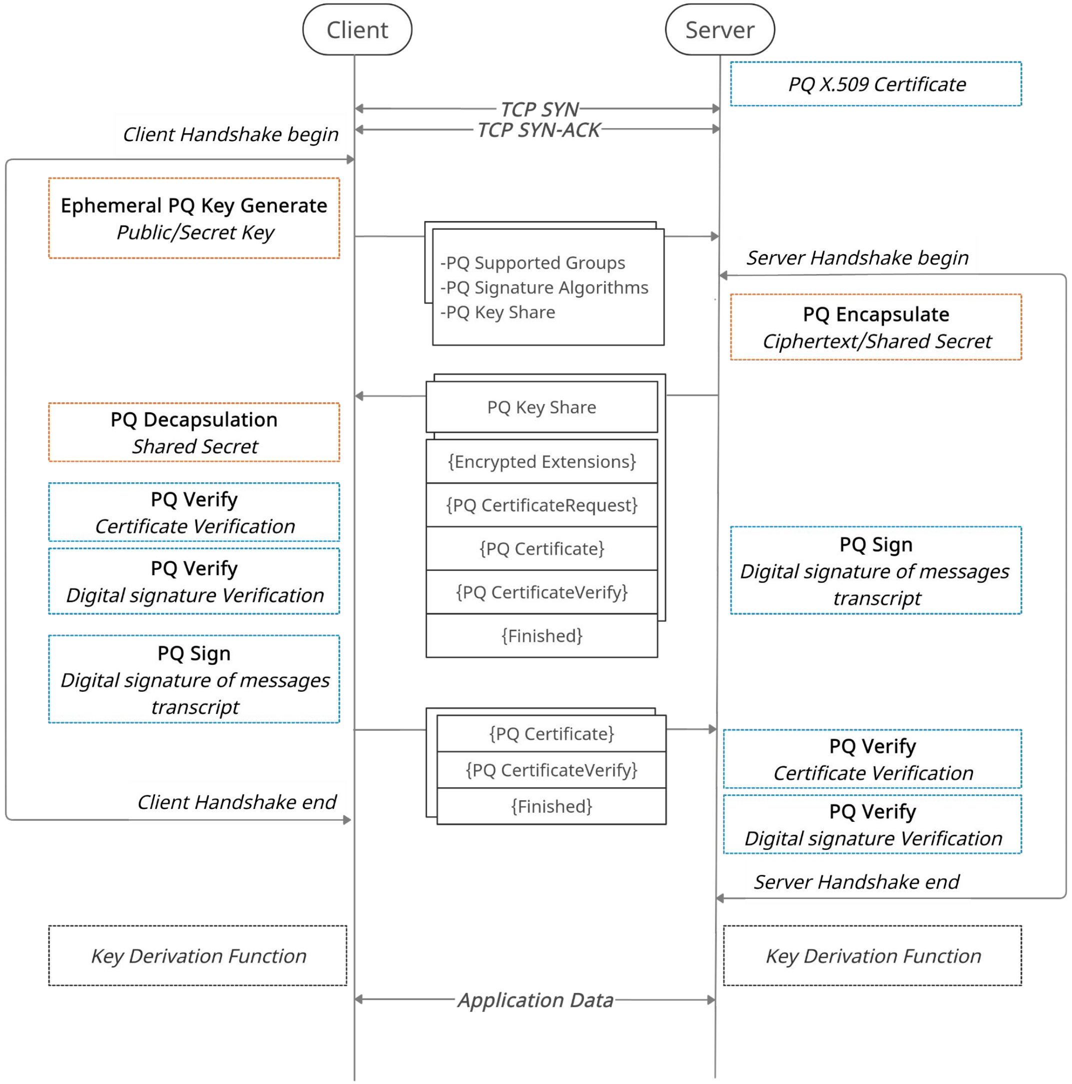
Architectural Changes (3rd change)

- Introduce *Post-quantum Certificates*
  - Fork of OpenSSL from the Open Quantum Safe project
  - Produce digital certs with PQ algorithms
  - Introduce a base “Certificate Authority”, self-signed
  - Produce certificates for server and client, directly signed by the CA.



Client				**Server**
ClientHello {+KeyShare}	**Key Gen**	→		
				ClientHello ServerHello {+KeyShare}
ServerHello *{+KeyShare}*	**Secret Gen**	←	**Key Gen** **Secret Gen**	
		←		EncryptedExtensions
EncryptedExtensions		←		CertificateRequest
CertificateRequest		←		Certificate
Certificate	**Verify**	←	**Sign**	CertificateVerify
CertificateVerify	**Verify**	←		Finished
Finished		←		[ApplicationData]
[Application Data]				
Certificate		→		
CertificateVerify	**Sign**	→	**Verify**	Certificate
Finished		→	**Verify**	CertificateVerify
[Application Data]		→		*Finished*
				[Application Data]
Application Data		↔		Application Data

Architectural Changes



Client

Server

PQ X.509 Certificate

TCP SYN

TCP SYN-ACK

Client Handshake begin

Ephemeral PQ Key Generate
Public/Secret Key

-PQ Supported Groups
-PQ Signature Algorithms
-PQ Key Share

Server Handshake begin

PQ Encapsulate
Ciphertext/Shared Secret

PQ Key Share

{Encrypted Extensions}

{PQ CertificateRequest}

{PQ Certificate}

{PQ CertificateVerify}

{Finished}

PQ Decapsulation
Shared Secret

PQ Verify
Certificate Verification

PQ Verify
Digital signature Verification

PQ Sign
Digital signature of messages transcript

PQ Sign
Digital signature of messages transcript

{PQ Certificate}

{PQ CertificateVerify}

{Finished}

PQ Verify
Certificate Verification

PQ Verify
Digital signature Verification

Client Handshake end

Server Handshake end

Key Derivation Function

Key Derivation Function

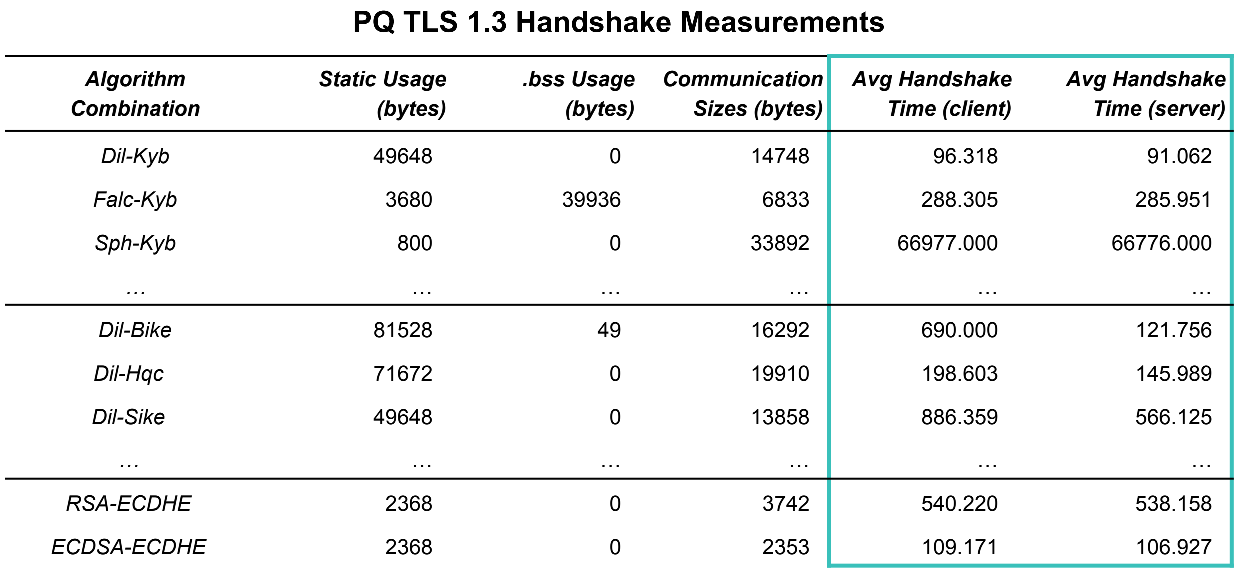
Application Data

Experimental Equipment

- *NUCLEO F439ZI evaluation board*
  - 32-bit ARM Cortex-M4 at 180 MHz
  - 192 KB of usable RAM
  - 2 MB of Flash memory
- *PC*
  - Intel i7-1165G7, 8 cores at 2.8 GHz
- *Access Point*
  - connected with Ethernet with mean RTT 0.493 ms
- *Oscilloscope*
  - PicoScope 5444B
  - Sampling rate of 5 MS/s for 5 seconds window

PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

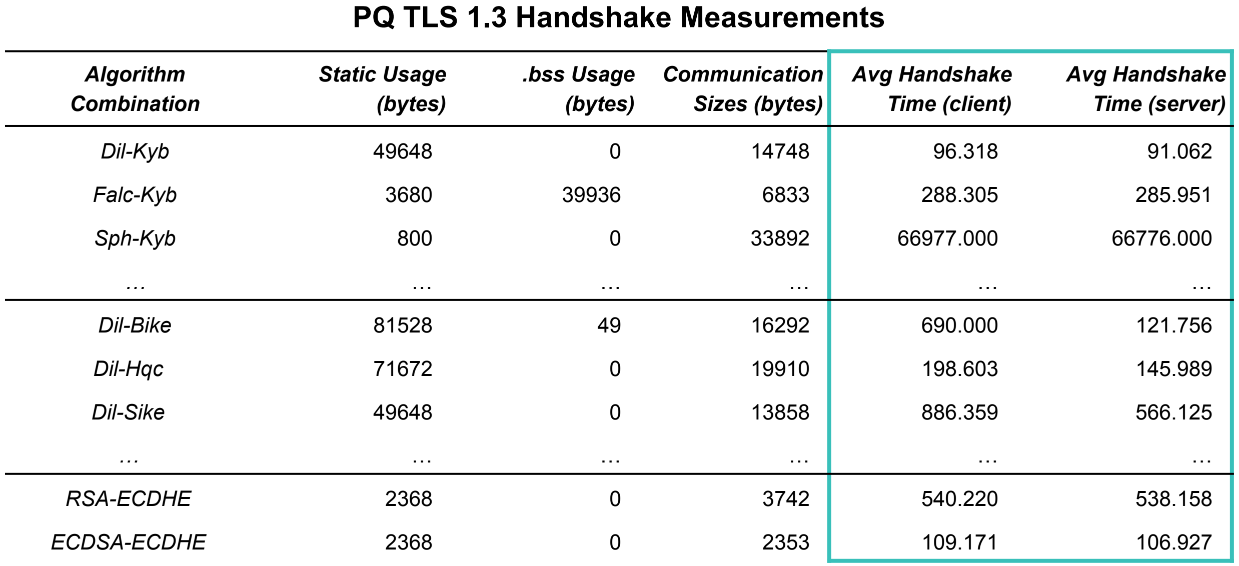


PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Average Handshake Time

- *Dil-Kyb* performs very good and better than *Falc-Kyb*
- *Sph-Kyb* is unusable (fast version is unusable in terms of memory)
- *Dil-Bike* as a client performs bad, as a server good
- *Dil-Hqc* performs good (better than *Falc-Kyb*)
- *Dil-Sike* performs poorly
- *RSA-ECDHE* performs poorly but *ECDSA-ECDHE* very good
  - *Dil-Kyb* is faster



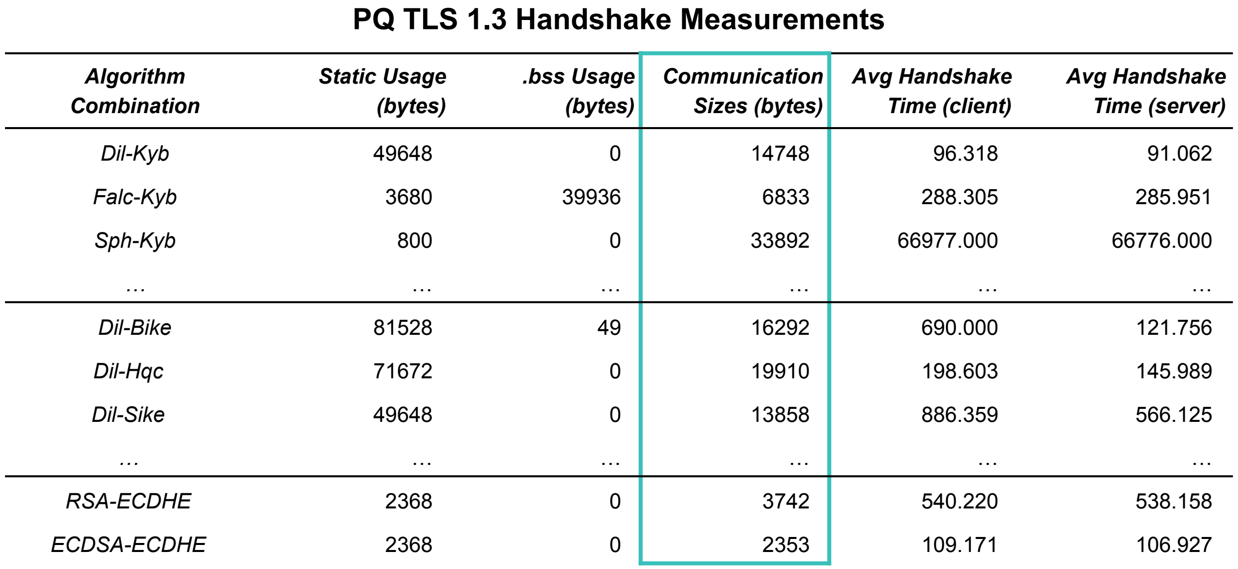
PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Average Handshake Time

- Notes:
  - *Optimized* version of PQ algorithms are very fast (sometimes faster than traditional)
  - May be furtherly optimised in the future
    - Employ better software (Cortex-M4 assembly)
    - Employ hardware

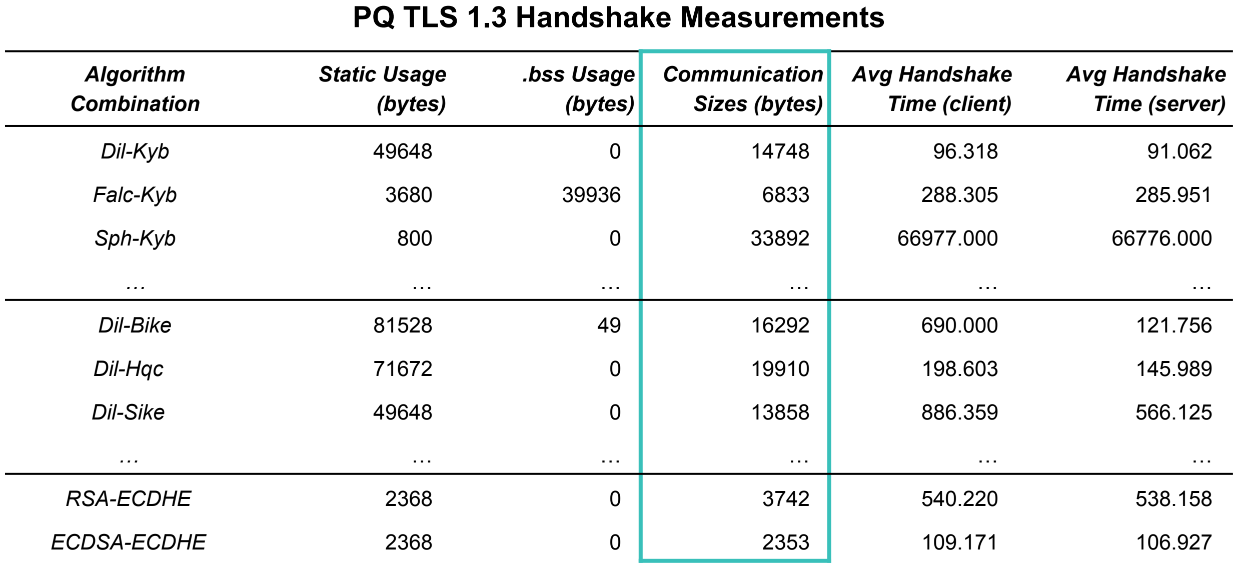
- Communication Sizes



PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Performance regarding *bandwidth*
- Reminder: Communication Size = Total bytes *sent* **and** *received* by a peer during the TLS handshake
- Dominated by the Authentication sizes (certificates and digital signatures)
  - Even more in our mutual authentication scenario
  - Although some times KEM sizes affect them

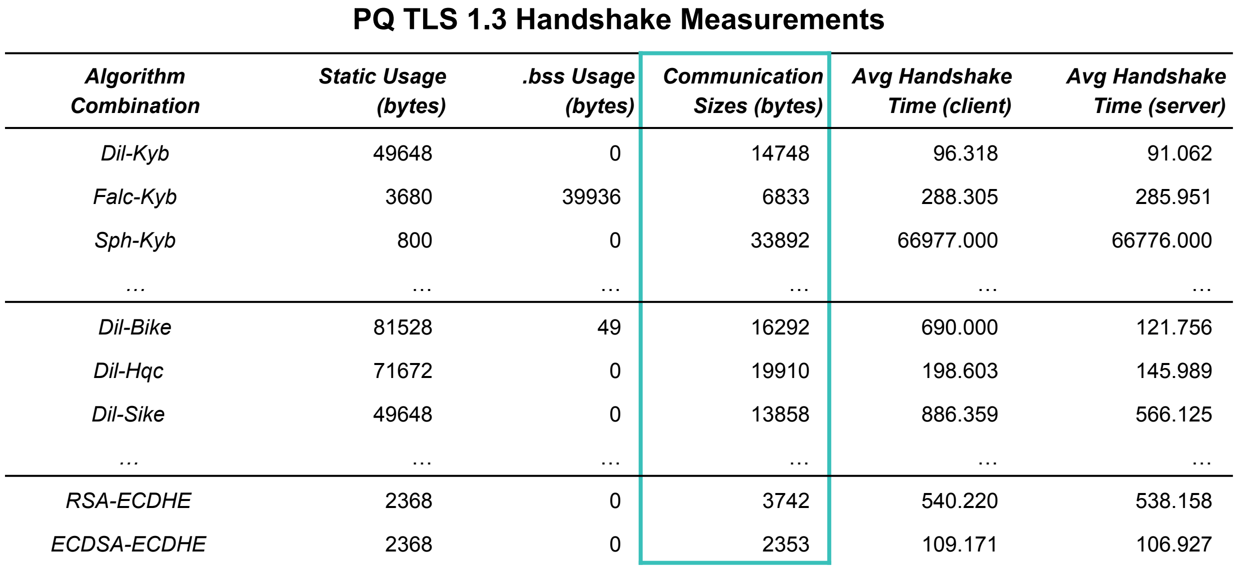


PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Communication Sizes

- *Falc-Kyb* uses ~7 KB of bandwidth
- *Dil-Kyb* uses ~2 times the bandwidth than *Falc-Kyb*
- *Sph-Kyb* uses ~5 times the bandwidth than *Falc-Kyb*
- *Dil-Bike* and *Dil-Hqc* use more bandwidth than *Falc-Kyb*
- *Dil-Sike* has the smallest communication sizes
- Traditional has an order of magnitude smaller sizes

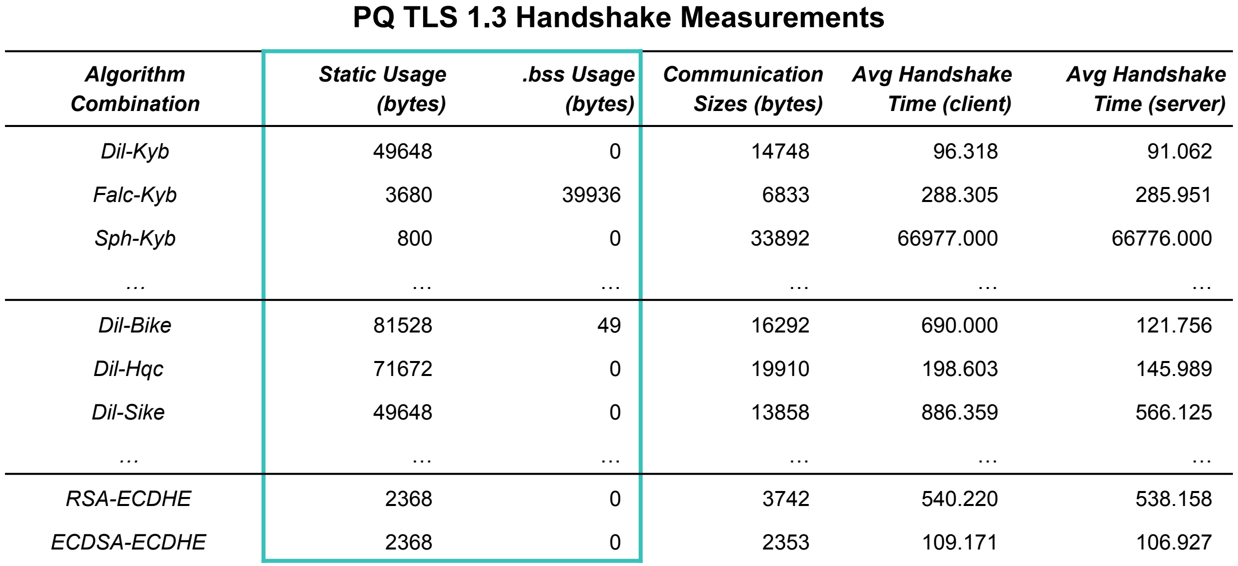


PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Communication Sizes

- Notes:
  - Post-quantum introduces larger sizes (in the order of 10 KB)
  - Can’t be reduced in the future
  - Can’t employ hardware or optimized software
  - Main disadvantage of post-quantum algorithms

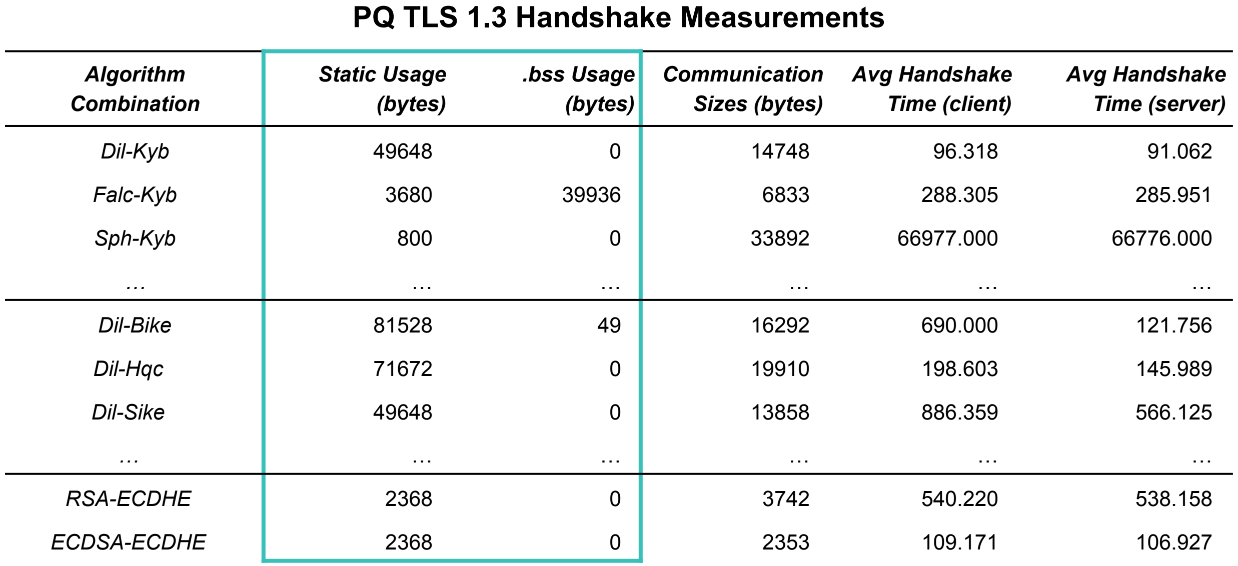


PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Memory Requirements

- RAM requirements is decisive in resource-constrained embedded systems
- Total board memory: 192 KB
- *Dil-Kyb* and *Falc-Kyb* consumes ~25% of total memory
- *Sph-Kyb* uses merely 800 bytes
- *Dil-Bike*, *Dil-Hqc* and *Dil-Sike* uses from 25%-41% of total memory
- *Traditional* combinations uses very little memory



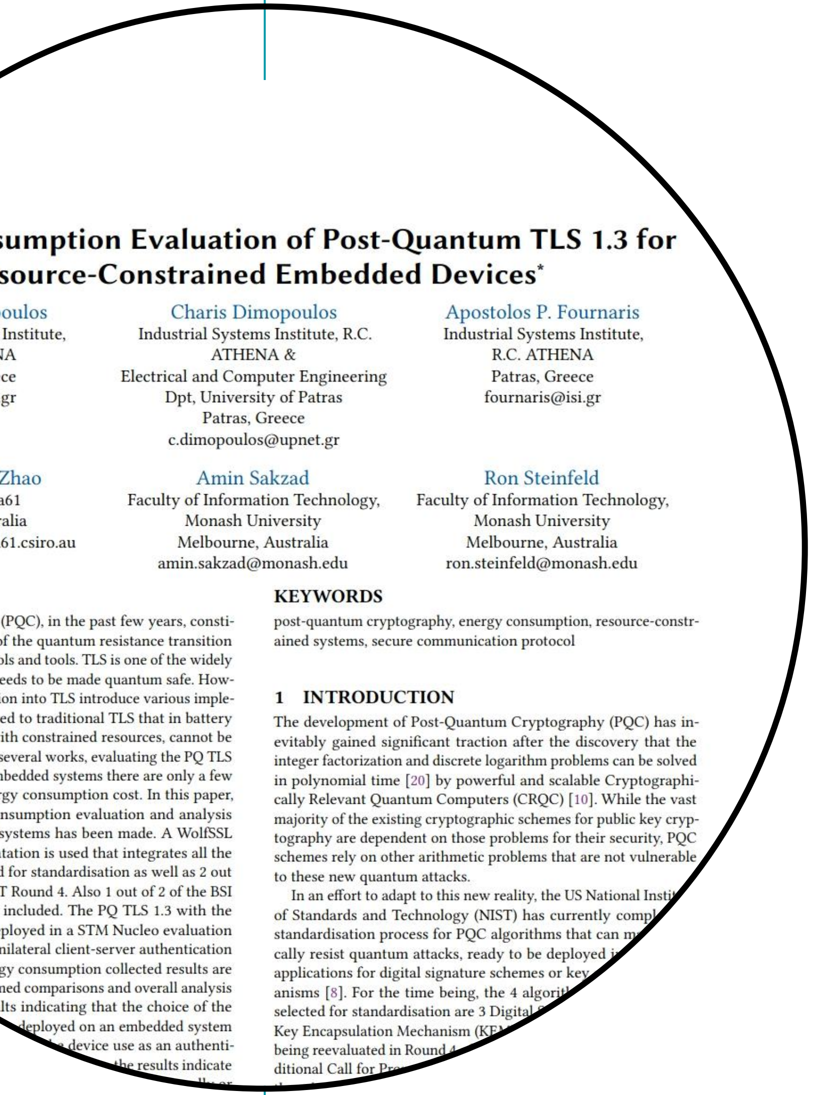
PQ TLS 1.3 Handshake Measurements

Algorithm Combination	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (client)*	*Avg Handshake Time (server)*
Dil-Kyb	49648	0	14748	96.318	91.062
Falc-Kyb	3680	39936	6833	288.305	285.951
Sph-Kyb	800	0	33892	66977.000	66776.000
…	…	…	…	…	…
Dil-Bike	81528	49	16292	690.000	121.756
Dil-Hqc	71672	0	19910	198.603	145.989
Dil-Sike	49648	0	13858	886.359	566.125
…	…	…	…	…	…
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

- Memory Requirements

- Notes:
  - This problem is better shown in higher security levels
    - Dilithium5
  - Some algorithms can’t fit at all
    - Classic McEllice (4th round candidate)
  - Maybe not a problem; memory optimized versions in the future

Extending our previous work



sumption Evaluation of Post-Quantum TLS 1.3 for
source-Constrained Embedded Devices*

oulos
Institute,
NA
ce
gr

Charis Dimopoulos
Industrial Systems Institute, R.C.
ATHENA &
Electrical and Computer Engineering
Dpt, University of Patras
Patras, Greece
c.dimopoulos@upnet.gr

Apostolos P. Fournaris
Industrial Systems Institute,
R.C. ATHENA
Patras, Greece
fournaris@isi.gr

Zhao
a61
alia
61.csiro.au

Amin Sakzad
Faculty of Information Technology,
Monash University
Melbourne, Australia
amin.sakzad@monash.edu

Ron Steinfeld
Faculty of Information Technology,
Monash University
Melbourne, Australia
ron.steinfeld@monash.edu

(PQC), in the past few years, consti-
of the quantum resistance transition
ols and tools. TLS is one of the widely
eeds to be made quantum safe. How-
on into TLS introduce various imple-
ed to traditional TLS that in battery
ith constrained resources, cannot be
several works, evaluating the PQ TLS
nbedded systems there are only a few
rgy consumption cost. In this paper,
nsumption evaluation and analysis
systems has been made. A WolfSSL
tation is used that integrates all the
d for standardisation as well as 2 out
T Round 4. Also 1 out of 2 of the BSI
included. The PQ TLS 1.3 with the
ployed in a STM Nucleo evaluation
nilateral client-server authentication
gy consumption collected results are
ned comparisons and overall analysis
lts indicating that the choice of the
deployed on an embedded system
device use as an authenti-
the results indicate

KEYWORDS

post-quantum cryptography, energy consumption, resource-constrained systems, secure communication protocol

1 INTRODUCTION

The development of Post-Quantum Cryptography (PQC) has inevitably gained significant traction after the discovery that the integer factorization and discrete logarithm problems can be solved in polynomial time [20] by powerful and scalable Cryptographically Relevant Quantum Computers (CRQC) [10]. While the vast majority of the existing cryptographic schemes for public key cryptography are dependent on those problems for their security, PQC schemes rely on other arithmetic problems that are not vulnerable to these new quantum attacks.

In an effort to adapt to this new reality, the US National Insti
of Standards and Technology (NIST) has currently comp
standardisation process for PQC algorithms that can m
cally resist quantum attacks, ready to be deployed i
applications for digital signature schemes or key
anisms [8]. For the time being, the 4 algorit
selected for standardisation are 3 Digital
Key Encapsulation Mechanism (KE
being reevaluated in Round 4
ditional Call for Pr

- PQC algorithms from ***pqm4***:
  - All NIST ***PQC algorithms*** selected for standardisation
  - 2 out of 3 from NIST PQC Round 4
  - 1 out of 2 of the BSI recommendations
- Integration in ***WolfSSL's*** impl. of TLS 1.3
- Benchmarked them on a board with ***Cortex-M4***
- Energy and power measurements
- Interesting results…

The source code is publicly available as two repositories at:
https://gitlab.com/g_tasop/pq-wolfssl-for-embedded
https://gitlab.com/g_tasop/pq-wolfssl-for-pc

Experimental Calculations

- Nucleo board **IDD jumper** for connecting an Ammeter in series
- We add a small **“shunt” resistor** (1.5 Ohm) in the jumper
- Oscilloscope - **differential Voltage** across the “shunt” resistor
- Knowing the Voltage we calculate the **Power consumption**:
  - $P = V * I = V * (V / R) = V^2 / R$
- Knowing the avg Time of a handshake we calculate the average **Energy Consumption**:
  - $E_{avg} = P * t_{avg}$

Experimental Scenarios

Two most typical IoT scenarios:

Mutual authentication scenario:

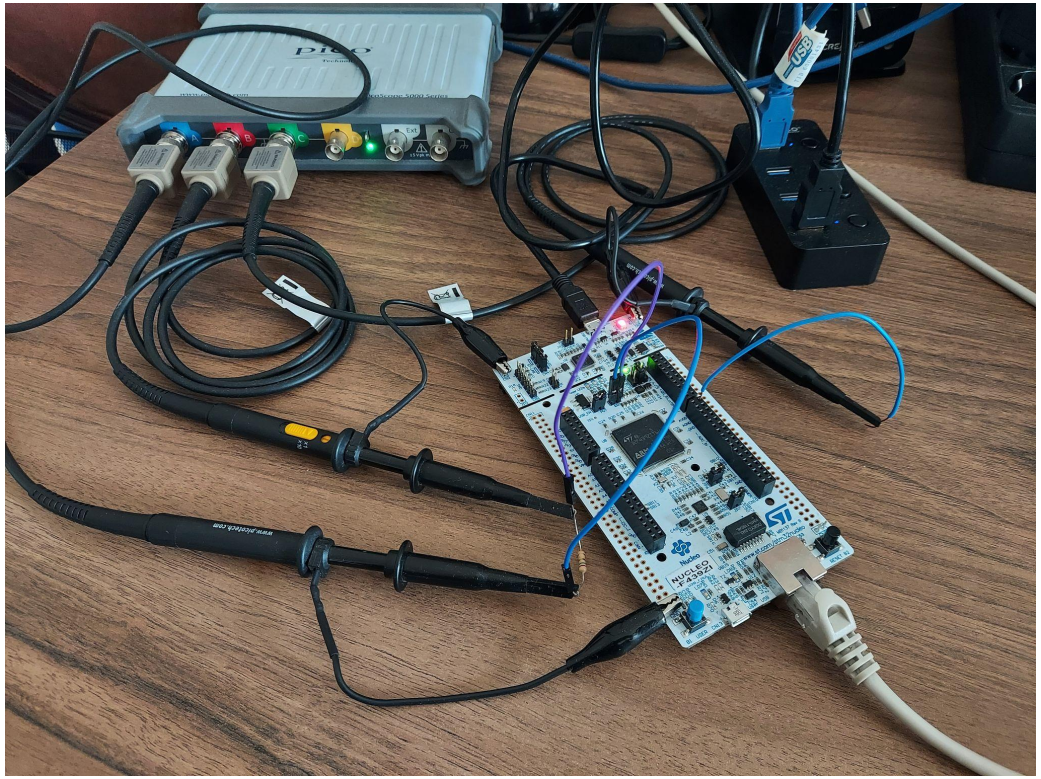
- An end-node is connected to another device (an end-node or a powerful device)
- Both are authenticated (e.g MQTT)
- Evaluated **TLS client** and **TLS server**

Unilateral authentication scenario:

- An end-node is connected to a more powerful device (server)
- Only the server is authenticated (sensor-cloud communication)
- Evaluated only **TLS client**

Experimental Setup

Our experimental setup:



Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			***Energy (mJ)***		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			*Energy (mJ)*		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Algorithm Combinations

- Power consumption is not the same between PQ combinations - not linear with time
- Depends on the operations of each PQ algorithm
- Specific on Cortex-M4 MCUs - not on higher x86 CPUs [2]

[2] **Mobile Energy Requirements of the Upcoming NIST Post-Quantum Cryptography Standards**. Markku-Juhani O. Saarinen. 2020. In 2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). 23–30. https://doi.org/10.1109/MobileCloud48802.2020.00012

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			***Energy (mJ)***		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 1st Scenario

- Client and Server in the mutual authentication scenario perform similarly
- Except BIKE
  - Highly asymmetric KEM operations
- Also HQC - smaller difference

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			***Energy (mJ)***		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 1st Scenario

- *Dil+Kyb* performs the best
- *Falc+Kyb* consumes more energy - still good performance
- *Sph+Kyb* needs more than 11 Joule - extremely high energy consumption
- *Dil+Bike* on **server** has a competitive energy consumption
- *Dil+Hqc* also has a very good performance - better than *Falc+Kyb*
- *FrodoKEM* combination cannot fit in the board in a mutual authentication scenario

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			*Energy (mJ)*		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 1st Scenario

- All of PQC combinations perform better than ***RSA+ECDHE***, except *Sph+Kyb* and *Dil+Bike* as a client
- *Dil+Kyb* performs better than very efficient ***ECDSA+ECDHE***

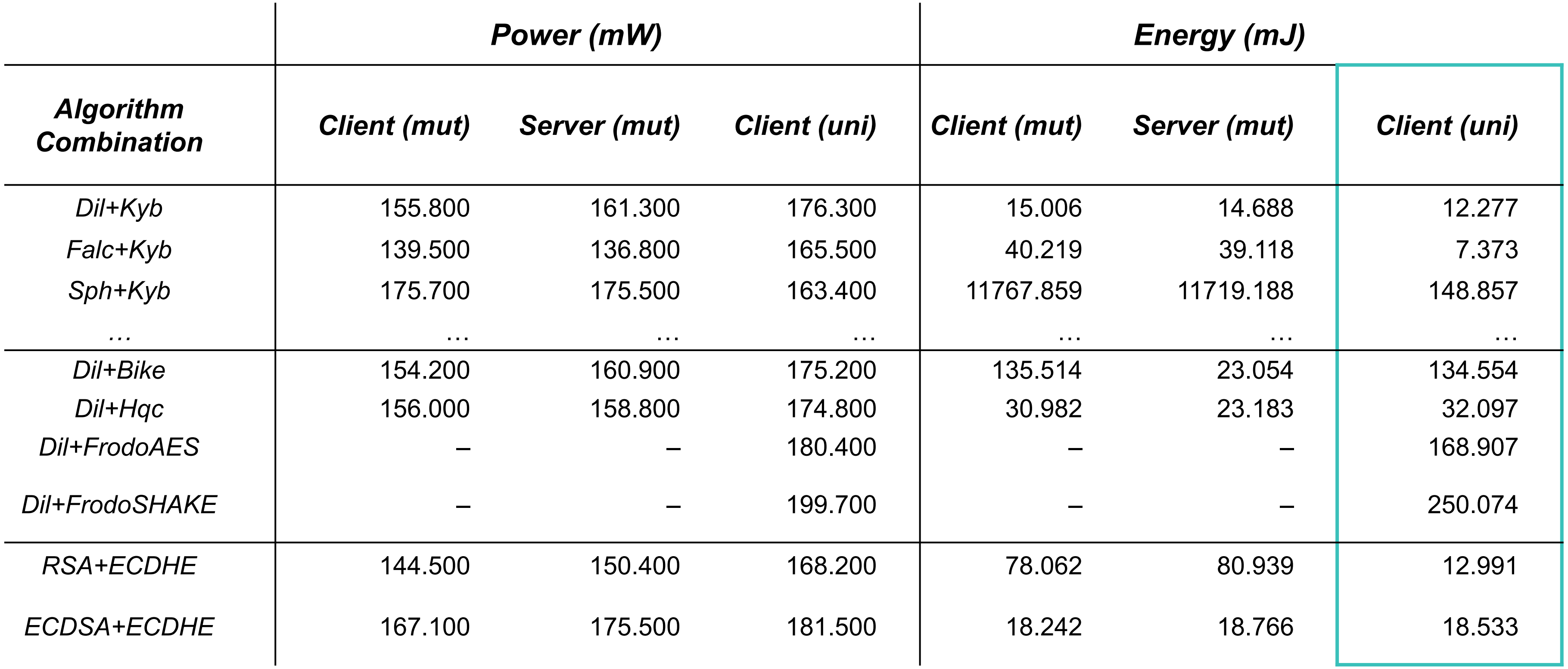
Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			Energy (mJ)		
Algorithm Combination	*Client (mut)*	*Server (mut)*	*Client (uni)*	*Client (mut)*	*Server (mut)*	*Client (uni)*
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 2nd Scenario

- Only TLS client evaluation on this scenario - TLS server would be the same as Server (mut)
- *Dil+Kyb* performs well
- *Falc+Kyb* performs better!
- *Sph+Kyb* is now usable
  - Higher energy consumption but its a conservative choice

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake



	Power (mW)			*Energy (mJ)*		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 2nd Scenario

- *Dil+Bike* is costly - Bike’s TLS client operations are costly
- *Dil+Hqc* performs well
- It is now viable to measure FrodoKEM
    - Good performance for a conservative choice
        - AES variant is more efficient
        - Can be pushed further with AES co-processors

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			Energy (mJ)		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
…	…	…	…	…	…	…
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Comparison for 2nd Scenario

- Comparison with *Traditional* algorithm combinations
- *Falc+Kyb* is more efficient than **RSA+ECDHE** and **ECDSA+ECDHE**
    - Almost twice as efficient
- *Dil+Kyb* is also marginally more efficient than **RSA+ECDHE** and **ECDSA+ECDHE**
- *Dil+Hqc* performs good - but needs x2 times more energy
- The rest of the algorithms require a lot more energy

Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake

	Power (mW)			***Energy (mJ)***		
Algorithm Combination	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil+Kyb	155.800	161.300	176.300	15.006	14.688	12.277
Falc+Kyb	139.500	136.800	165.500	40.219	39.118	7.373
Sph+Kyb	175.700	175.500	163.400	11767.859	11719.188	148.857
Dil3+Kyb3	161.600	164.200	178.400	25.392	25.203	17.569
Falc5+Kyb3	137.100	133.900	168.900	81.505	78.875	11.190
Dil3+Kyb5	161.200	162.900	180.100	26.693	24.848	19.066
Falc5+Kyb5	138.00	133.700	172.500	83.052	79.191	12.759
Dil+Bike	154.200	160.900	175.200	135.514	23.054	134.554
Dil+Hqc	156.000	158.800	174.800	30.982	23.183	32.097
Dil+FrodoAES	–	–	180.400	–	–	168.907
Dil+FrodoSHAKE	–	–	199.700	–	–	250.074
RSA+ECDHE	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	167.100	175.500	181.500	18.242	18.766	18.533

- Full Table

- *Falc+Kyb* (uni) requires LESS energy in high security levels than ***RSA+ECDHE***

Conclusions

- PQ TLS 1.3 in resource-constrained embedded systems:
    - Can be **very fast** (even on low resources)
    - But suffer from **large communication sizes**
    - Also, requires a **lot of memory**
        - Some security levels can’t fit
        - Some algorithms can’t fit
        - Some may be optimized in the future

Conclusions

- We can get an energy efficiency upgrade from the Post-Quantum transition!

- Dil+Kyb in a **mutual authentication** scenario

- Falc+Kyb (and marginally Dil+Kyb) in a **unilateral authentication** scenario

- However in more energy expensive communication channels (e.g GSM) the overall picture could be different.

- *Extra*: Security upgrade (traditional AND post-quantum) with *Falc+Kyb* (on unilateral authentication scenario) as higher security levels are still more energy efficient than traditional TLS 1.3

Future Research

- Hardware Agility
  - Efforts on linear algebra arithmetic of lattice-based crypto:
    - Use RSA coprocessor (Bos et al., 2020)
    - Configurable (i.e. supporting different parameters) hardware accelerator on FPGA (Derya et al., 2021)
  - **Question:** How to design **crypto-agile**, **high-performance**, **side-channel resistant** PQC Hardware Security Module (HSM)?

Future Research

- **Efficient** side-channel resistant implementation:
  - Existing NIST PQC implementations are **constant-time** (i.e. **timing/cache** side-channel resistant).
  - However, IoT requires protections of more side-channels:
    - Power, electromagnetic, fault, …
  - Countermeasures are **expensive**:
    - E.g. Migliore et al. (2019) showed speed overhead of masking (countermeasure against power analysis) Dilithium:
      - 5.66x/7.8x/13.4x of Order-1/2/3 masking of KeyGen
      - 5.68x/11.77x/28.3x of Order-1/2/3 masking of Sign
  - **Question:** Develop side-channel countermeasures with **lower** overhead.

Future Research

- Broader use case scenarios e.g.
  - Vehicle communication (Bindel et al., 2022)
  - Satellite communication
  - 6G communication
- Many has specific protocol/standard with performance requirements (bandwidth, latency, etc.)
- **Question:** How to do PQC migration for these use cases?
- CSIRO is involved in PQC for 6G!

Thank You!

References

Nouri Alnahawi, Nicolai Schmitt, Alexander Wiesmaier, Andreas Heinemann, Tobias Grasmeyer: On the State of Crypto-Agility. IACR Cryptol. ePrint Arch. 2023: 487 (2023)

Nina Bindel, Sarah McCarthy, Geoff Twardokus, Hanif Rahbari: Drive (Quantum) Safe! -- Towards Post-Quantum Security for V2V Communications. IACR Cryptol. ePrint Arch. 2022: 483 (2022)

Joppe W. Bos, Joost Renes, Christine van Vredendaal: Post-Quantum Cryptography with Contemporary Co-Processors: Beyond Kronecker, Schönhage-Strassen & Nussbaumer. USENIX Security Symposium 2022: 3683-3697

Kemal Derya, Ahmet Can Mert, Erdinç Öztürk, Erkay Savas: CoHA-NTT: A Configurable Hardware Accelerator for NTT-based Polynomial Multiplication. Microprocess. Microsystems 89: 104451 (2022)

Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, Ko Stoffelen: pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. IACR Cryptol. ePrint Arch. 2019: 844 (2019)

Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, Pierre-Alain Fouque: Masking Dilithium - Efficient Implementation and Side-Channel Evaluation. ACNS 2019: 344-362

Christian Paquin, Douglas Stebila, Goutam Tamvada: Benchmarking Post-quantum Cryptography in TLS. PQCrypto 2020: 72-91

Smyslov, V. (2014). Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation (No. rfc7383).

Smyslov, V. (2022). RFC 9242: Intermediate Exchange in the Internet Key Exchange Protocol Version 2 (IKEv2).

Douglas Stebila, Michele Mosca: Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. SAC 2016: 14-37

Stebila, D., Fluhrer, S., & Gueron, S. (2020). Hybrid key exchange in TLS 1.3. IETF draft.

George Tasopoulos, Jinhui Li, Apostolos P. Fournaris, Raymond K. Zhao, Amin Sakzad, Ron Steinfeld: Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems. ISPEC 2022: 432-451

George Tasopoulos, Charis Dimopoulos, Apostolos P. Fournaris, Raymond K. Zhao, Amin Sakzad, Ron Steinfeld: Energy Consumption Evaluation of Post-Quantum TLS 1.3 for Resource-Constrained Embedded Devices. IACR Cryptol. ePrint Arch. 2023: 506.

Tjhai, C. J., Tomlinson, M., Bartlett, G., Fluhrer, S., Van Geest, D., Garcia-Morchon, O., & Smyslov, V. (2023). RFC 9370: Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2).

Thomas H¨aner, Samuel Jaques, Michael Naehrig, Martin Roetteler and Mathias Soeken. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. PQCrypto 2020.

Chris Peikert, 6 March 2022, Post-Quantum Cryptography, University of Michigan, http://web.eecs.umich.edu/~cpeikert/pubs/slides-qip22-tutorial.pdf.

Shor, Peter W. "Algorithms for quantum computation: discrete logarithms and factoring." Proceedings 35th annual symposium on foundations of computer science. Ieee, 1994.

Grover, Lov K. "A fast quantum mechanical algorithm for database search." Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. 1996.