wolfSSL

Post-Quantum Cryptography Status

Anthony Hu
Senior Software Developer

The Hague, The Netherlands
December 2025

Session Agenda

1. The Algorithms
2. wolfSSL PQC History and Current Status
3. Benchmarking ECC vs ML-DSA and ML-KEM
4. wolfSSL PQC Readiness and Migration Efforts
5. What Others are Doing Together with wolfSSL
6. Barriers to Migration

Company Overview

Medical Devices
Automotive / ADAS / ECUs
Industrial Systems
Internet of Things
Aviation
Satellite
Appliances
Battlefield Communication
VoIP
Electric Grid
Federal Governments
Games
We Secure the Internet by Securing Data
Intelligent Systems
Sensors
Routers
Smart Energy
Databases
Mobile Phones
Cloud Services
Connected Home
Applications
Printers

wolfSSL Technology Partners

wolfSSL Open Source Partners

wolfSSL Products

Core Products

- **wolfCrypt** — Crypto Engine
- **wolfSSL** — Lightweight SSL/TLS
- **wolfMQTT** — Lightweight MQTT Client
- **wolfSSH** — Lightweight SSH Client/Server
- **wolfTPM** — Lightweight TPM 2.0 Library
- **wolfBoot** — Secure Bootloader
- **wolfSentry** — Embedded IDPS
- **wolfEntropy** — SP800-90B Certifiable Software TRNG
- **wolfHSM** — wolfSSL for Hardware Security Modules

Certified Products

- *Dual-Licensing*
- *Delivered in source code*
- *Common API for FIPS 140-3 and DO-178C*

wolfCrypt FIPS 140-3/Validated

wolfCrypt DO-178C/DAL A

JSSE Provider | JCE Provider

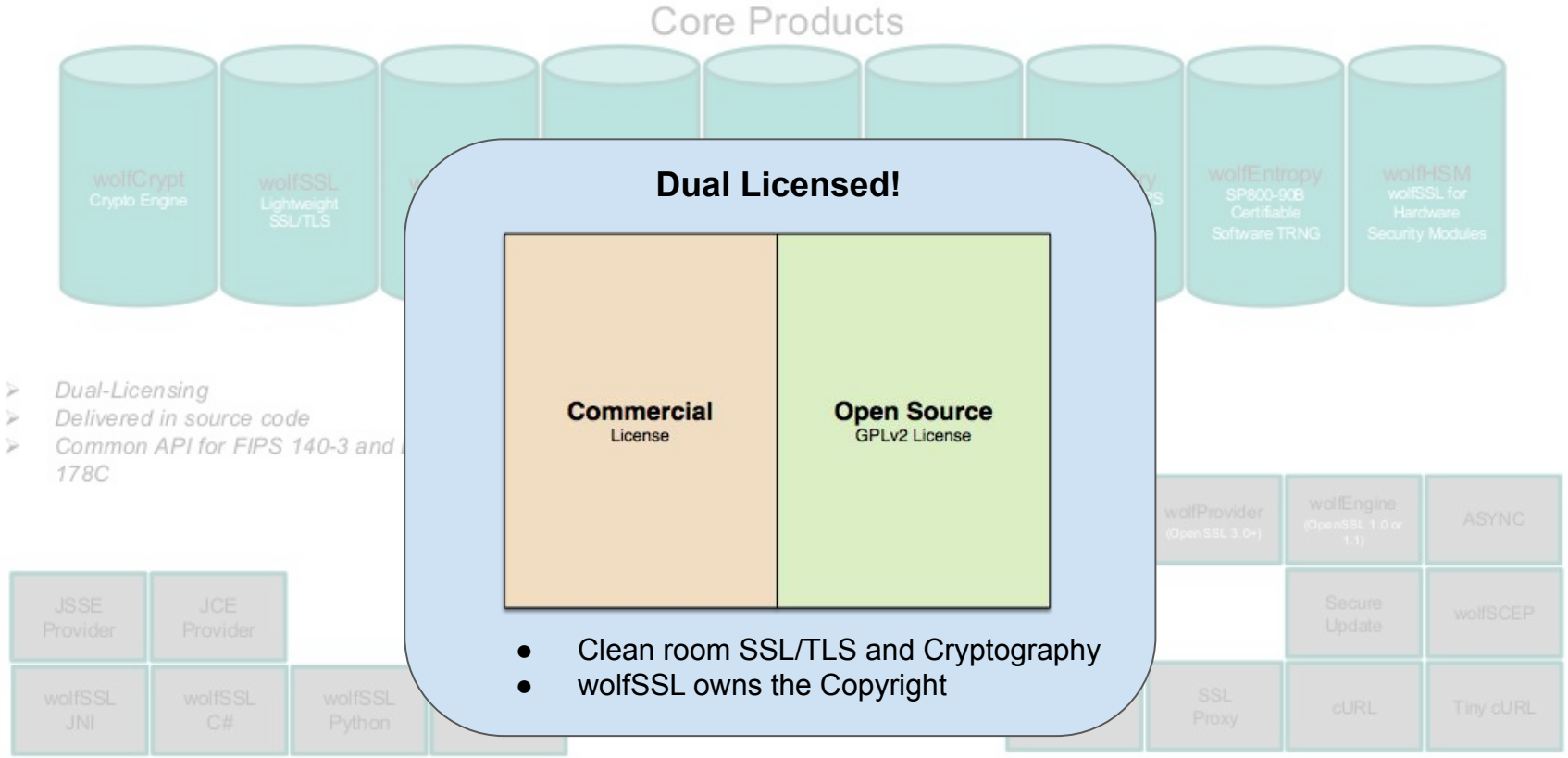
wolfSSL JNI | wolfSSL C# | wolfSSL Python | Javascript

wolfProvider (OpenSSL 3.0+) | wolfEngine (OpenSSL 1.0 or 1.1) | ASYNC

Secure Update | wolfSCEP

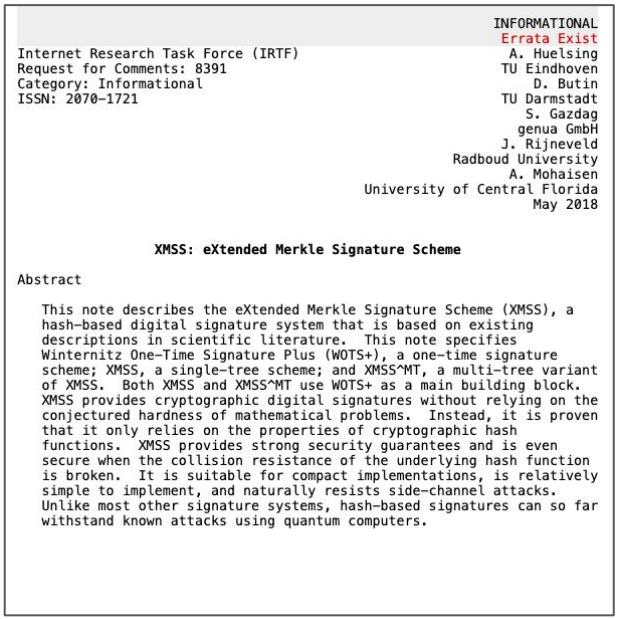
wolfCLU | SSL Proxy | cURL | Tiny cURL

wolfSSL Products



Hash-Based Signatures

- **NIST** standardized SLH-DSA in FIPS-205 standard document
- **IETF** standardized both of the following Stateful Hash-Based Signature algorithms:
  - **XMSS** (RFC 8391, 2018) - eXtended Merkle Signature Scheme
  - **LMS** (RFC 8554, 2019) - Leighton-Micali Hash-Based Signatures



INFORMATIONAL
Errata Exist

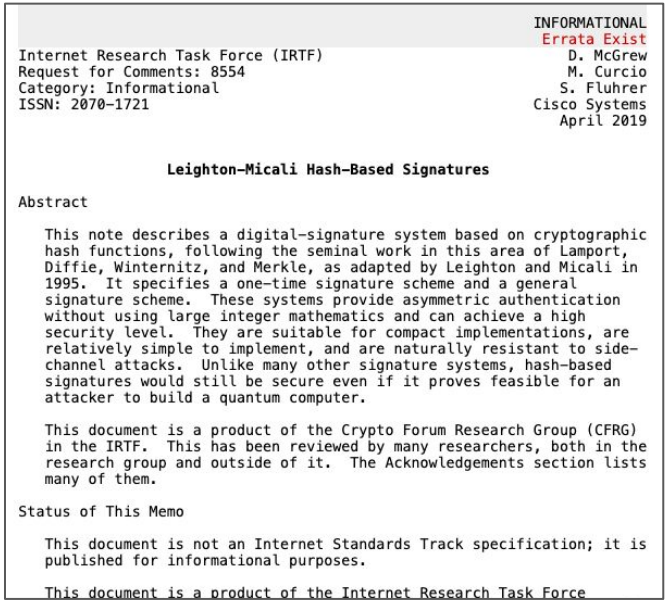
Internet Research Task Force (IRTF)
Request for Comments: 8391
Category: Informational
ISSN: 2070-1721

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gazdag
genua GmbH
J. Rijneveld
Radboud University
A. Mohaisen
University of Central Florida
May 2018

XMSS: eXtended Merkle Signature Scheme

Abstract

This note describes the eXtended Merkle Signature Scheme (XMSS), a hash-based digital signature system that is based on existing descriptions in scientific literature. This note specifies Winternitz One-Time Signature Plus (WOTS+), a one-time signature scheme; XMSS, a single-tree scheme; and XMSS^MT, a multi-tree variant of XMSS. Both XMSS and XMSS^MT use WOTS+ as a main building block. XMSS provides cryptographic digital signatures without relying on the conjectured hardness of mathematical problems. Instead, it is proven that it only relies on the properties of cryptographic hash functions. XMSS provides strong security guarantees and is even secure when the collision resistance of the underlying hash function is broken. It is suitable for compact implementations, is relatively simple to implement, and naturally resists side-channel attacks. Unlike most other signature systems, hash-based signatures can so far withstand known attacks using quantum computers.



INFORMATIONAL
Errata Exist

Internet Research Task Force (IRTF)
Request for Comments: 8554
Category: Informational
ISSN: 2070-1721

D. McGrew
M. Curcio
S. Fluhrer
Cisco Systems
April 2019

Leighton-Micali Hash-Based Signatures

Abstract

This note describes a digital-signature system based on cryptographic hash functions, following the seminal work in this area of Lamport, Diffie, Winternitz, and Merkle, as adapted by Leighton and Micali in 1995. It specifies a one-time signature scheme and a general signature scheme. These systems provide asymmetric authentication without using large integer mathematics and can achieve a high security level. They are suitable for compact implementations, are relatively simple to implement, and are naturally resistant to side-channel attacks. Unlike many other signature systems, hash-based signatures would still be secure even if it proves feasible for an attacker to build a quantum computer.

This document is a product of the Crypto Forum Research Group (CFRG) in the IRTF. This has been reviewed by many researchers, both in the research group and outside of it. The Acknowledgements section lists many of them.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Research Task Force

NIST Post-Quantum Cryptography Standardization

- **FIPS 203**
  - Specifies **ML-KEM**
  - Based on **CRYSTALS-KYBER**
  - Defines 3 parameter sets (ECDH sizes are in parenthesis for comparison)

Variant	Encapsulation Key	Decapsulation Key	Ciphertext	Shared Secret Key	AES Equivalence
ML-KEM-512	800 (64)	1632 (32)	768 (64)	32	128
ML-KEM-768	1184 (96)	2400 (48)	1088 (96)	32	192
ML-KEM-1024	1568 (131)	3168 (66)	1568 (131)	32	256

  - Appropriate for replacement of quantum-vulnerable **key exchange algorithms** (ex: ECDH, FFDH)
  - Performance is very good, cryptographic artifact sizes are larger than ECDH / FFDH

NIST Post-Quantum Cryptography Standardization

- **FIPS 204**
  - Specifies **ML-DSA**
  - Based on **CRYSTALS-Dilithium**
  - Defines 3 parameter sets (ECDSA sizes are in parenthesis for comparison)

Variant	Public Key	Private Key	Signature	AES Equivalence
ML-DSA-44	1312 (64)	2528 (32)	2420 (64)	128
ML-DSA-65	1952 (96)	4000 (48)	3293 (96)	192
ML-DSA-87	2592 (131)	4864 (66)	4595 (131)	256

  - Appropriate for replacement of quantum-vulnerable **digital signature algorithms** (ex: ECDSA, RSA)
  - Performance is on par, cryptographic artifact sizes are larger than ECDSA / RSA

wolfSSL

PQC History and Current Status

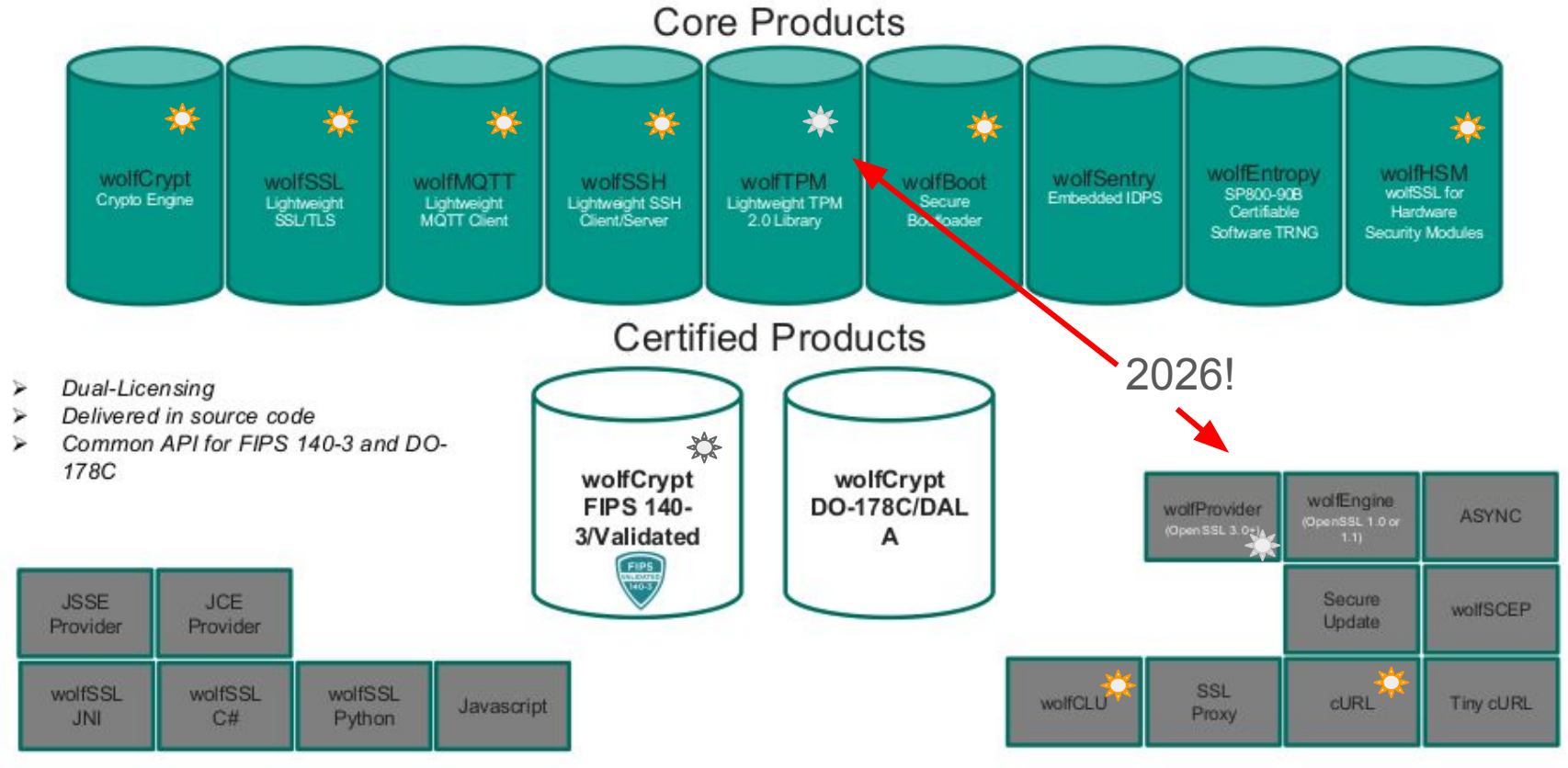
wolfSSL and Post-Quantum Cryptography

- **wolfSSL** has included **NTRU** support since **2010**!
  - We originally partnered with Security Innovation to bring experimental **NTRU** support to wolfSSL
    - **NTRU**
      - “**N**-th degree **T**runcated polynomial **R**ing **U**nits”
  - Deprecated and removed from wolfSSL as it is no longer being considered for standardization

wolfSSL and Post-Quantum Cryptography

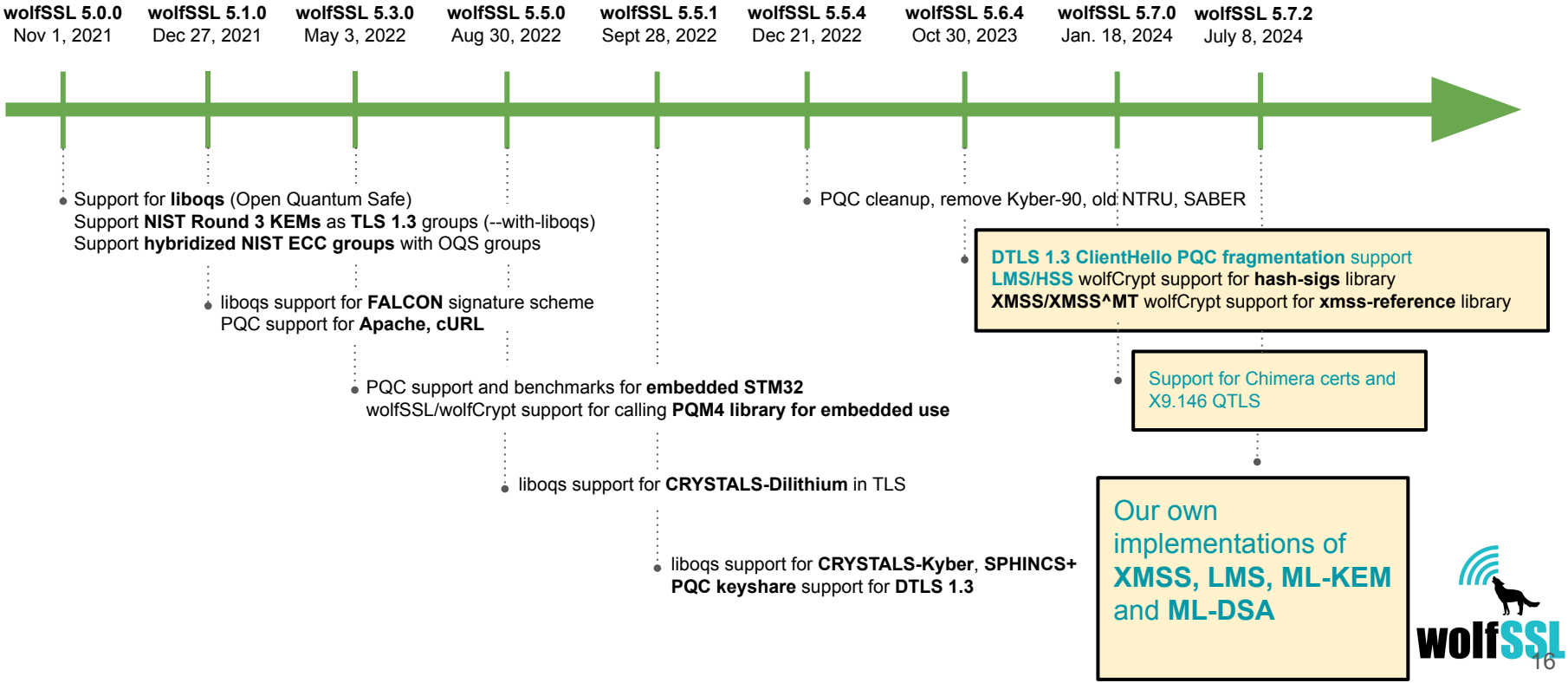
- In **2015**, wolfSSL added support for **QSH** (Quantum Safe Hybrid) handshake extension into TLS 1.2
  - https://tools.ietf.org/html/draft-whyte-qsh-tls12-02 ; expired
  - Deprecated and removed from wolfSSL

wolfSSL Products



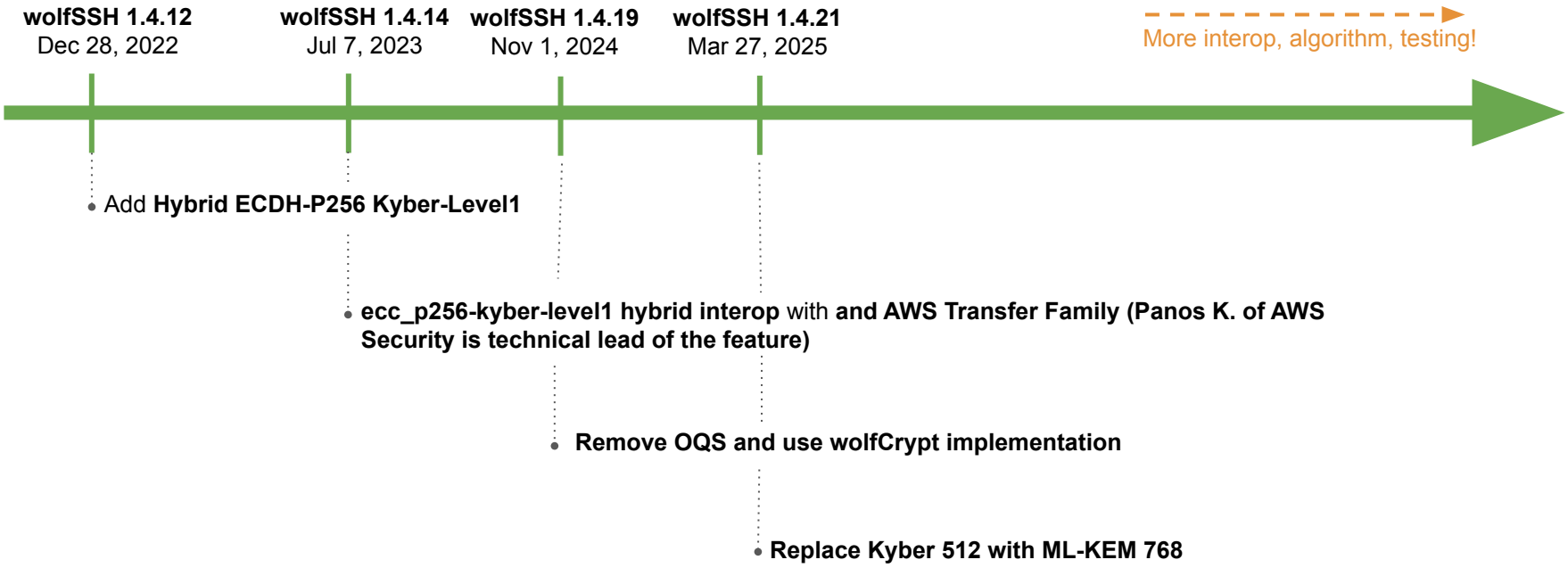
wolfSSL and Post-Quantum Cryptography

Flagship product. Cryptographic algorithm and TLS/DTLS protocol implementations.



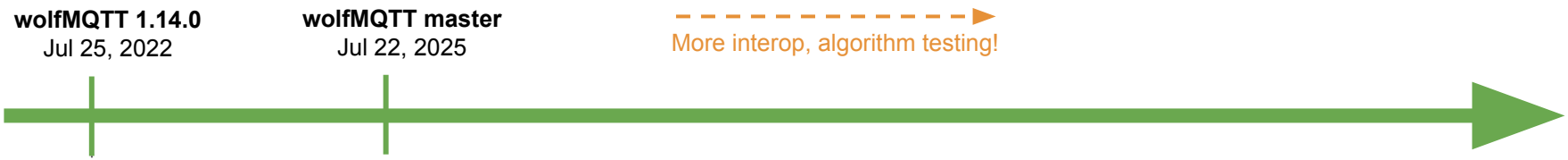
wolfSSH and Post-Quantum Cryptography

SSHv2 protocol implementation



wolfMQTT and Post-Quantum Cryptography

MQTTv5 publish and subscribe protocol implementation

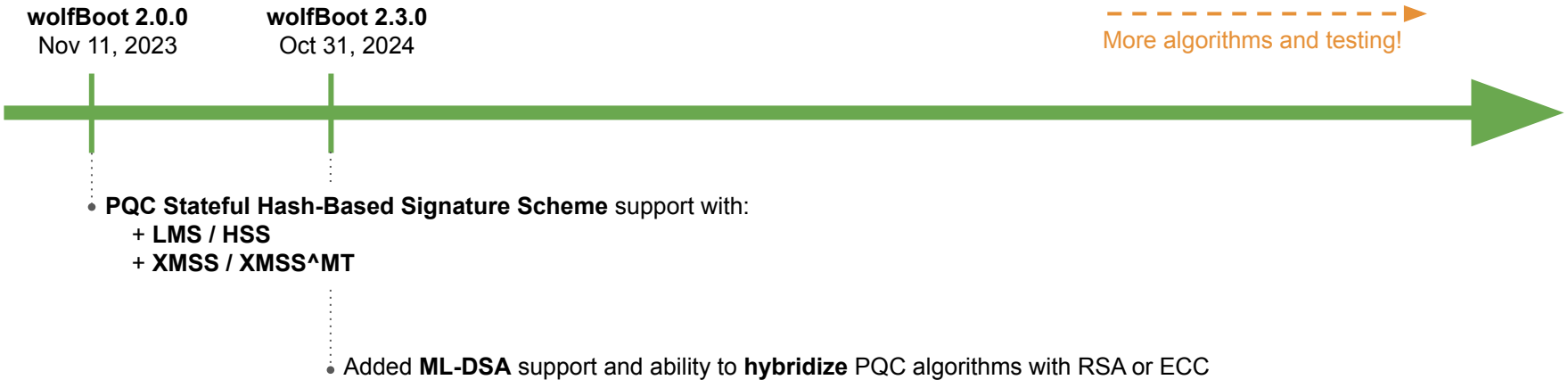


- **PQC** support for **KYBER_LEVEL_1, P256_KYBER_LEVEL1, FALCON_LEVEL1**
  Uses **liboqs** with **wolfSSL**
  Connects to **Mosquitto MQTT broker** integrated with OpenQuantumSafe project
  PQC with TLS 1.3 support for MQTT!

- Switch to ML_KEM_768 or SecP384r1MLKEM768 with ML-DSA-65 in TLS 1.3

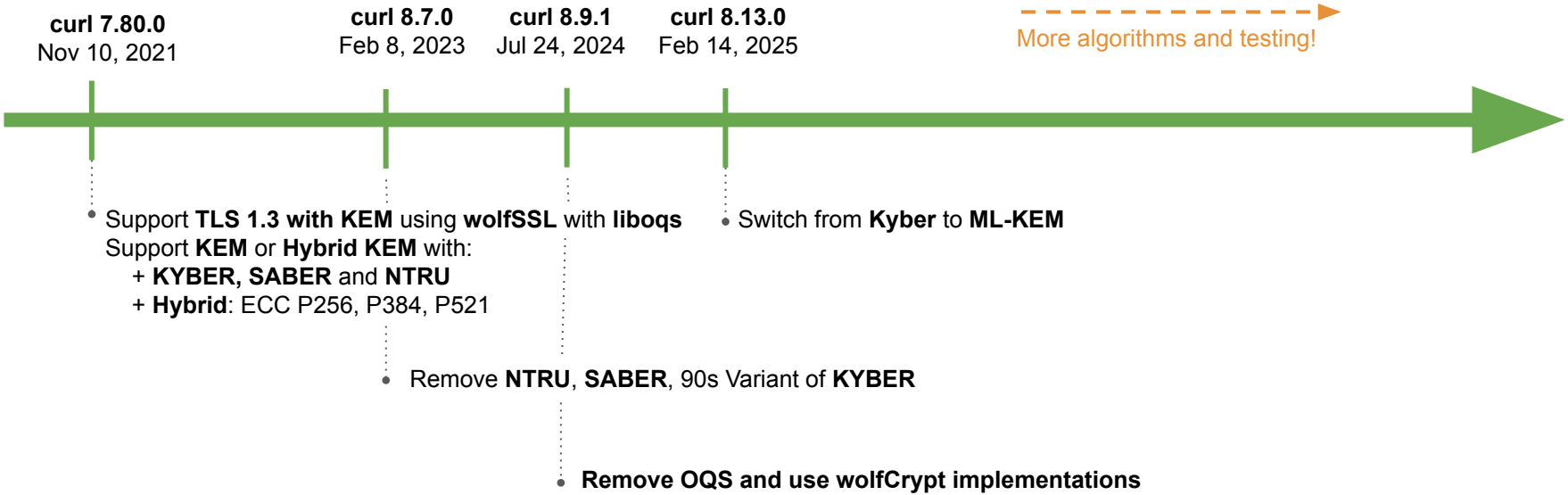
wolfBoot and Post-Quantum Cryptography

Fully featured low level boot-loader for firmware on embedded systems



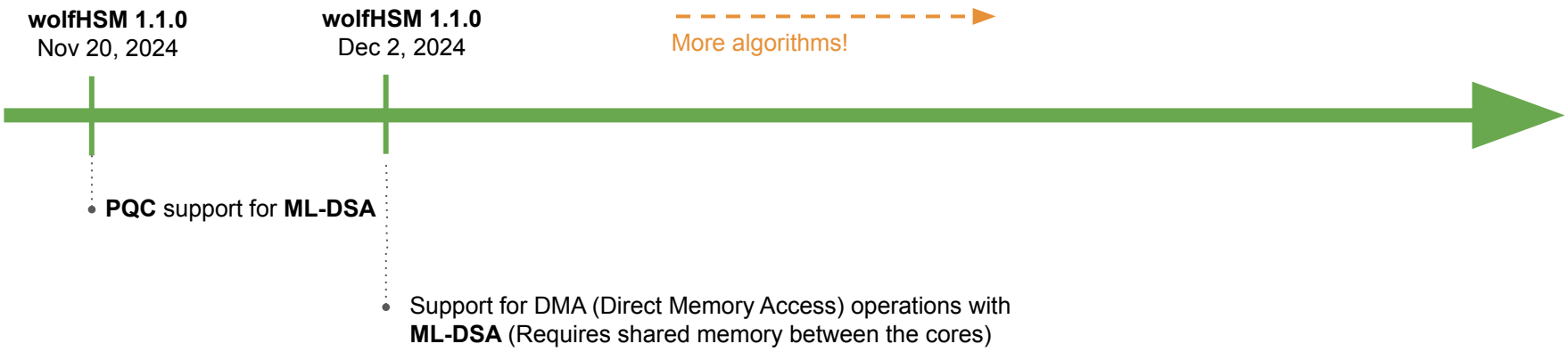
cURL and Post-Quantum Cryptography

Command-line tool and library, non-interactive client for many protocols



wolfHSM and Post-Quantum Cryptography

Software that runs on a single HSM core in a microcontroller.



Post-Quantum Roadmap

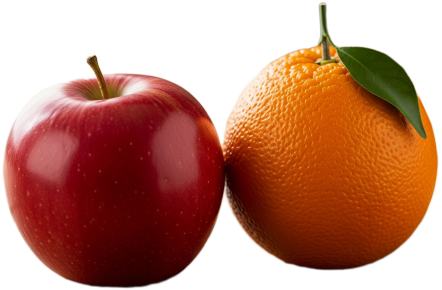
- wolfCrypt: McEliece, FrodoKEM, HQC, FN-DSA, SLH-DSA, tracking NIST's further standardization
- wolfCrypt-fips: Certification of ML-KEM, ML-DSA, LMS verification, XMSS verification
- wolfSSH: ML-KEM hybridization with X25519 and X448, ML-DSA certificate support
- wolfHSM: LMS and XMSS private key operations
- OSP: Porting PQConnect to an embedded system
- **Things our customers want and need**

Benchmarking ECC vs ML-DSA and ML-KEM

NIKEs Vs. KEMs

Bear with me please…

… the way the algorithm works dictates how you calculate your benchmarks!



Some Acronym Decoding

- **NIKE**
  - Non-Interactive Key Exchange
  - Examples: DH, ECDH
- **KEM**
  - Key Encapsulation Mechanism
  - Examples: RSA, ML-KEM

So What?

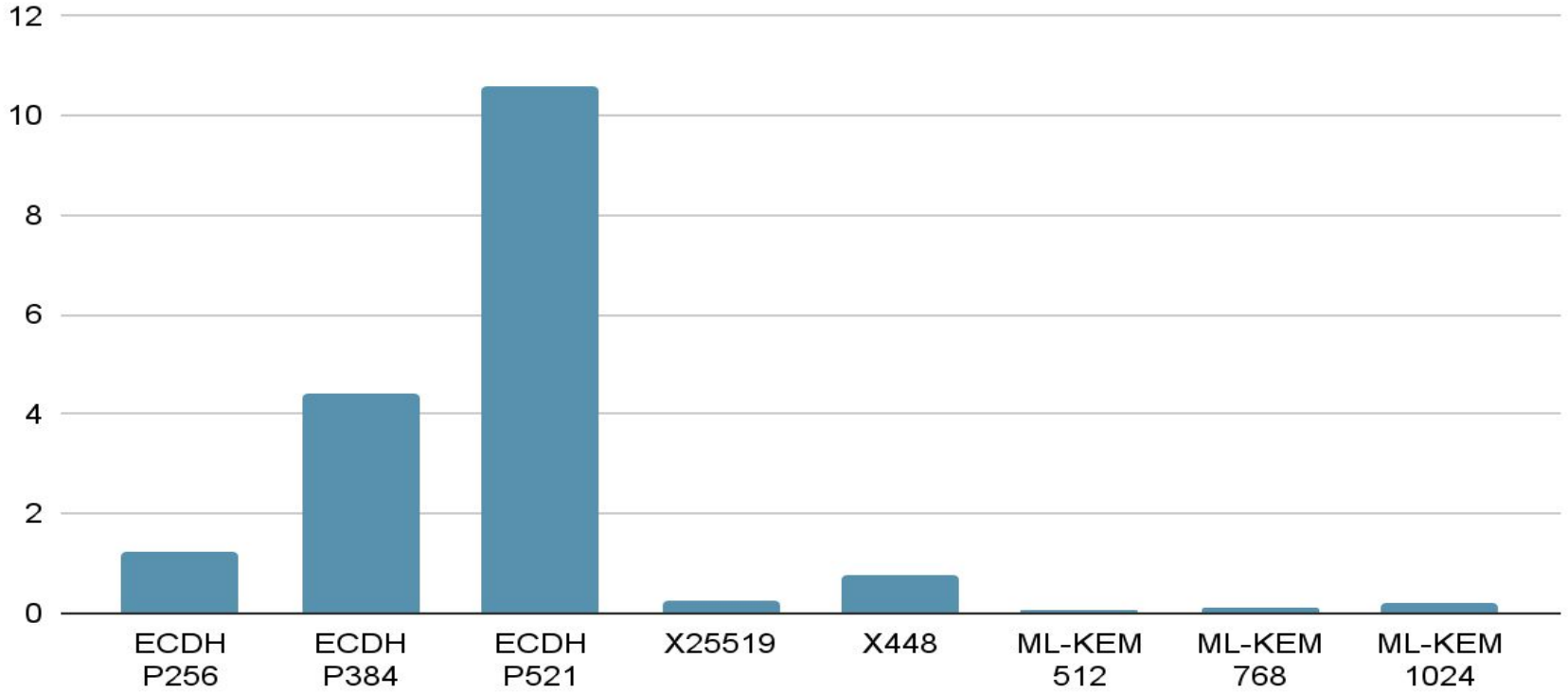
- In a NIKE, Keygen() and Derive() are each called **twice**
- In a KEM, Keygen(), Encapsulate() and Decapsulate() are each called **once**
- Therefore, when comparing the benchmarks for NIKEs and KEMs, you sum the time for both NIKE operations and then **double** it for a fair comparisons!

About These Benchmarks

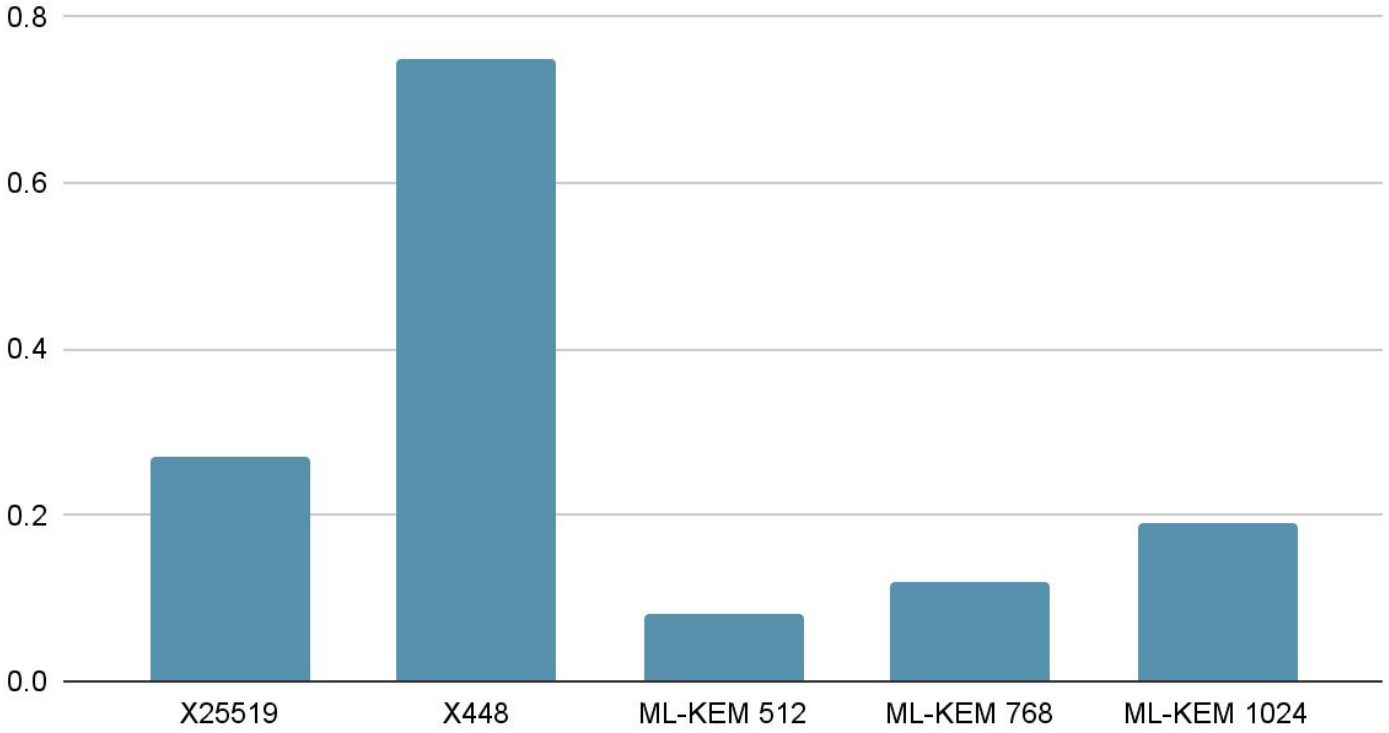
- The Machine: Lenovo Thinkpad L13 Yoga Gen 4
  - Memory: 16 GiB
  - Processor: AMD Ryzen 5 pro 7530u with radeon graphics × 12 (only 1 core used)
  - OS: Ubuntu 22.04.5 LTS (64-bit)
- WolfSSL/Wolfcrypt Library
  - Version: GIT Hash `5a8411a1ada39e93…` (November 12, 2025)
  - Configuration: (For assembly optimizations, add `--enable-all-asm`)

```
./configure --enable-mlkem --enable-dilithium \
    --enable-curve25519 --enable-curve448 \
    --enable-ed25519 --enable-ed448 \
    --enable-memory --enable-trackmemory=verbose --enable-stacksize=verbose
```

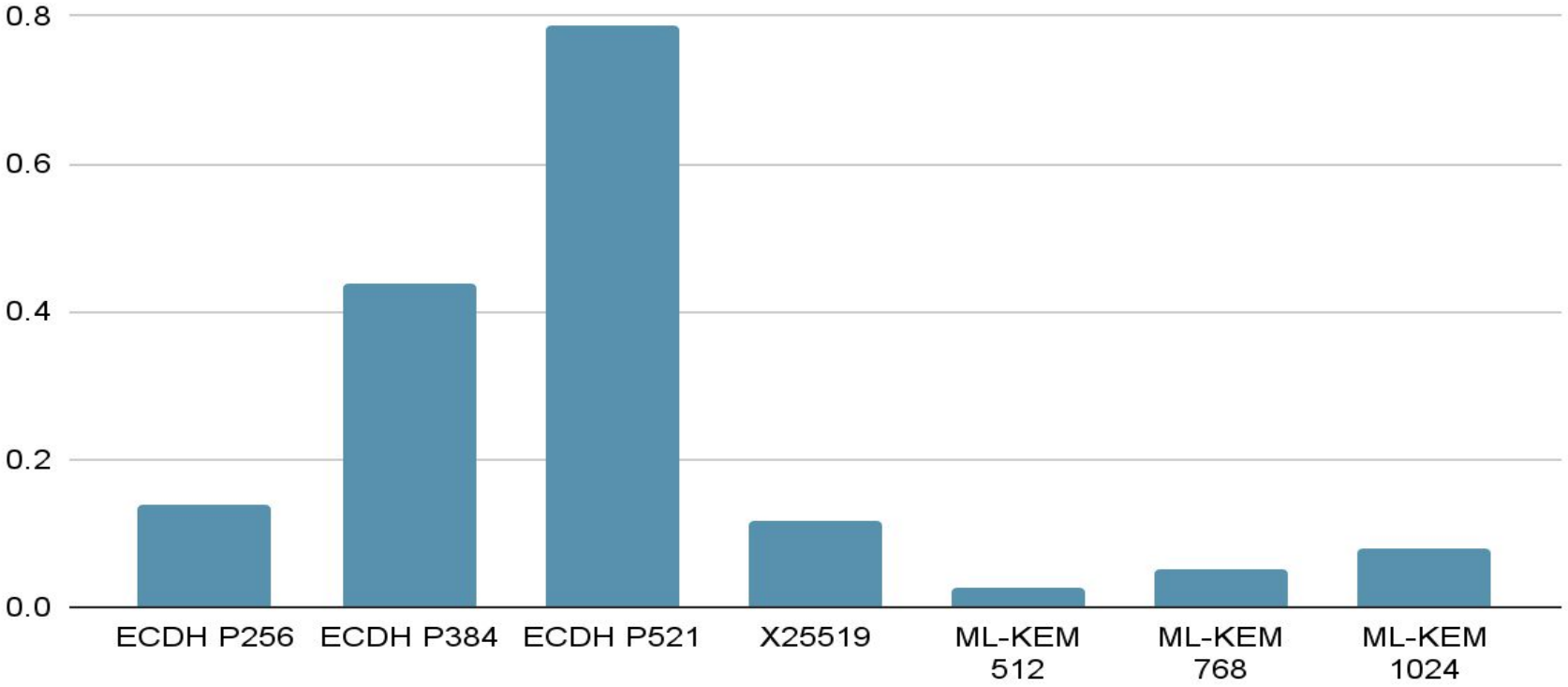
ECDH vs ML-KEM (ms) (NO Assembly Optimizations)



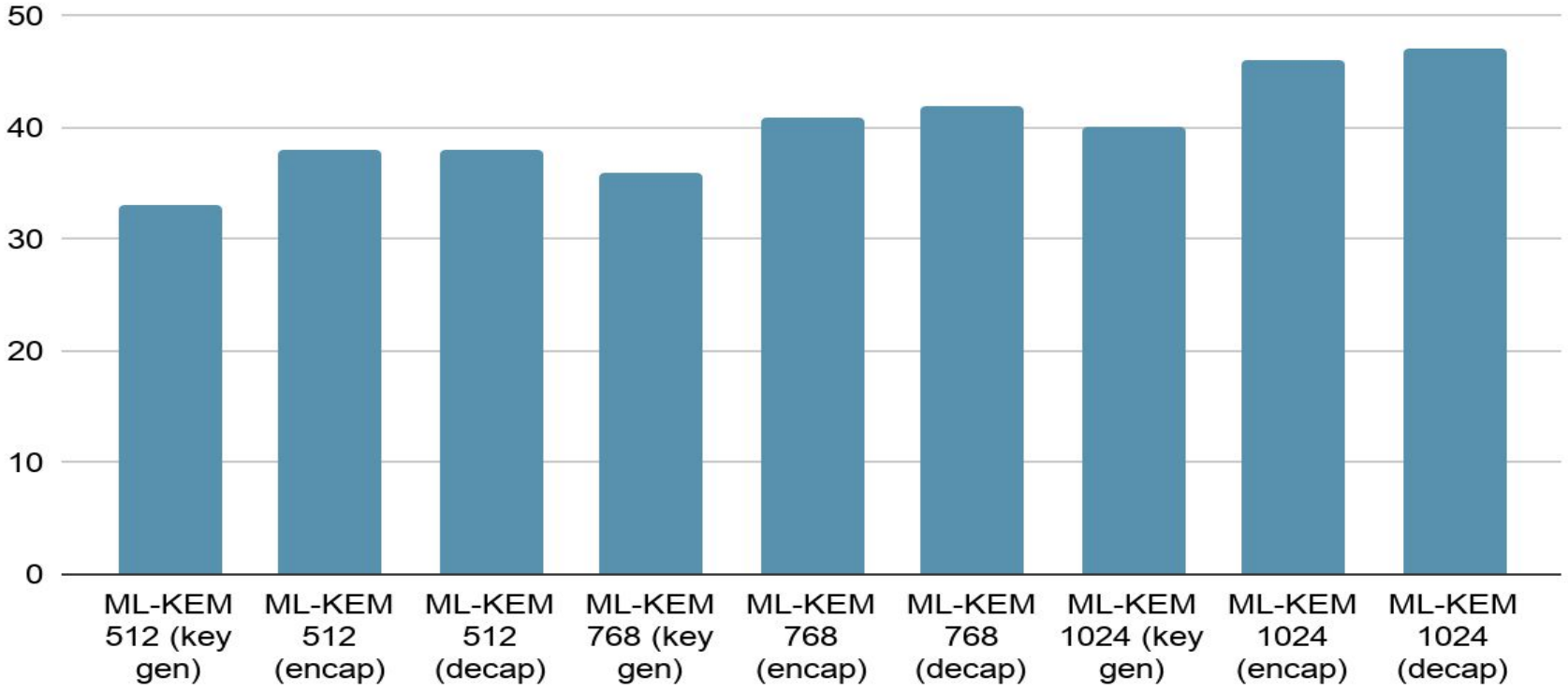
ECDH vs ML-KEM (ms) (NO Assembly Optimizations)



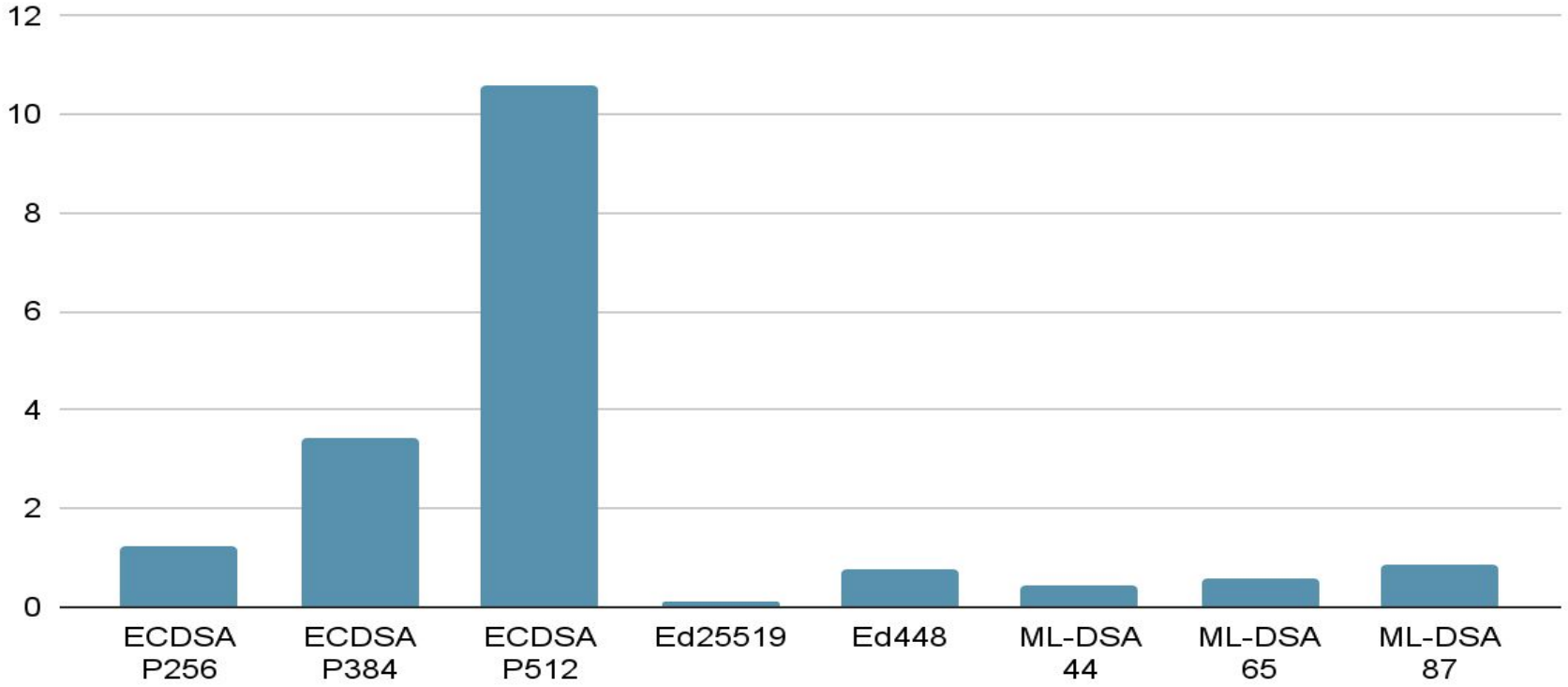
ECDH vs ML-KEM (ms) (WITH Assembly Optimizations)



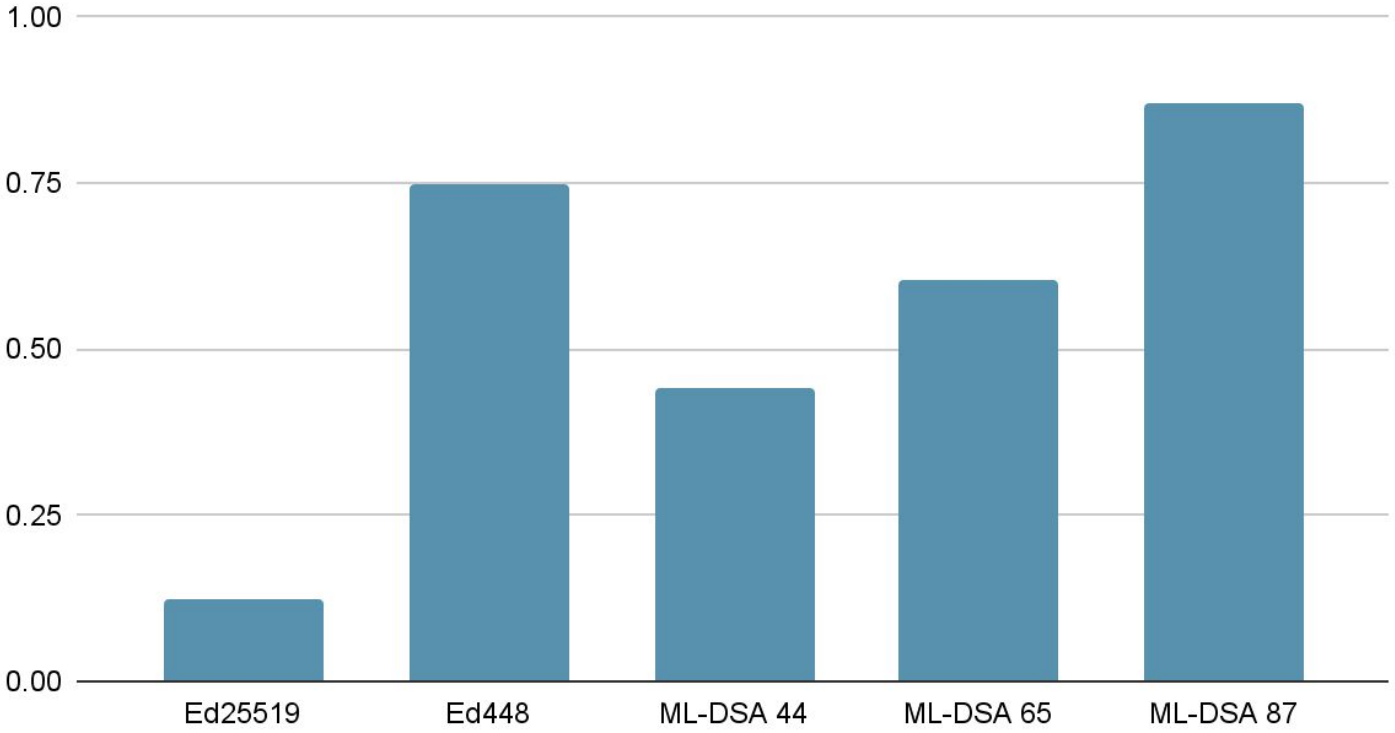
ML-KEM Memory Usage (Kilobytes)



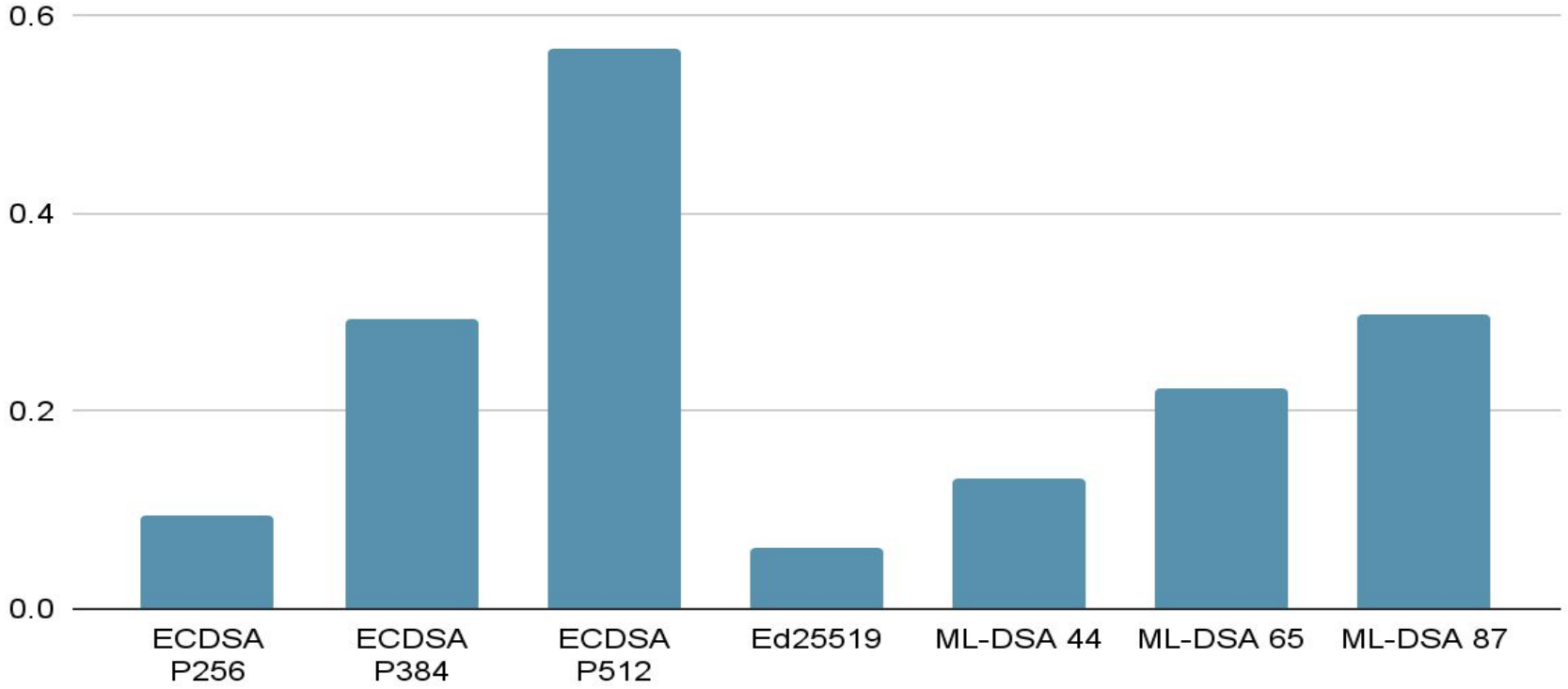
ECDSA vs ML-DSA (ms) (NO Assembly Optimizations)



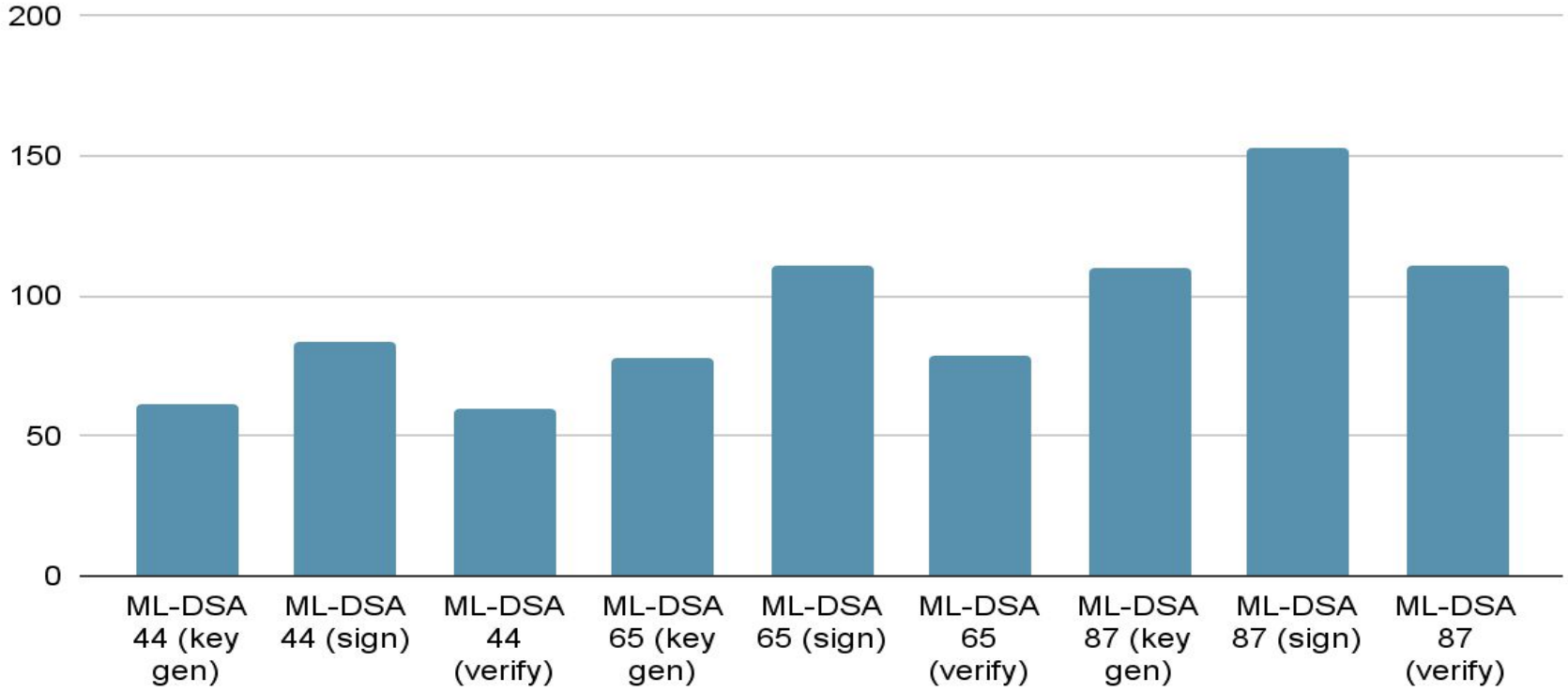
ECDSA vs ML-DSA (ms) (NO Assembly Optimizations)



ECDSA vs ML-DSA (ms) (WITH Assembly Optimizations)



ML-DSA Memory Usage (Kilobytes)



Raw Benchmarking CSV Data (NO Assembly Optimizations)

```
Algorithm,key size,operation,avg ms,ops/sec,heap_bytes,heap_allocs,stack_bytes,
ML-KEM 512 ,128,key gen,0.018,56342.399,3072,1,29848
ML-KEM 512 ,128,encap,0.024,41093.201,6656,1,30264
ML-KEM 512 ,128,decap,0.039,25944.774,7424,2,29088
ML-KEM 768 ,192,key gen,0.033,29911.428,6144,1,28608
ML-KEM 768 ,192,encap,0.037,27151.346,10752,1,29088
ML-KEM 768 ,192,decap,0.050,20112.430,11840,2,29088
ML-KEM 1024,256,key gen,0.047,21184.312,10240,1,28608
ML-KEM 1024,256,encap,0.057,17635.622,15872,1,29088
ML-KEM 1024,256,decap,0.086,11632.920,17440,2,29088
ECC   [      SECP256R1],256,key gen,0.461,2168.278,15640,5,12744
ECDHE [      SECP256R1],256,agree,0.452,2213.570,31280,10,25072
ECDSA [      SECP256R1],256,sign,0.436,2294.087,15640,5,27480
ECDSA [      SECP256R1],256,verify,0.342,2924.267,56818,20,21640
ECC   [      SECP224R1],224,key gen,0.531,1884.067,15640,5,12744
ECDHE [      SECP224R1],224,agree,0.516,1939.732,31280,10,25072
ECDSA [      SECP224R1],224,sign,0.493,2027.427,15640,5,27480
ECDSA [      SECP224R1],224,verify,0.358,2793.499,56818,20,21664
ECC   [      SECP384R1],384,key gen,1.145,873.408,15640,5,12744
ECDHE [      SECP384R1],384,agree,1.065,939.028,31280,10,25072
ECDSA [      SECP384R1],384,sign,1.404,712.039,15640,5,27608
ECDSA [      SECP384R1],384,verify,0.867,1152.959,56818,20,21960
ECC   [      SECP521R1],528,key gen,4.061,246.215,15640,5,12744
ECDHE [      SECP521R1],528,agree,3.777,264.780,31280,10,25072
ECDSA [      SECP521R1],528,sign,3.889,257.139,15640,5,27800
ECDSA [      SECP521R1],528,verify,2.647,377.761,56818,20,22464
CURVE,25519,key gen,0.068,14802.471,0,0,2488
CURVE,25519,agree,0.067,14926.161,0,0,3792
ED,25519,key gen,0.027,37608.794,0,0,2472
ED,25519,sign,0.029,35059.334,0,0,5120
ED,25519,verify,0.069,14475.545,0,0,6264
CURVE,448,key gen,0.192,5218.941,0,0,2472
CURVE,448,agree,0.183,5472.783,0,0,4160
ED,448,key gen,0.071,14044.977,0,0,2600
ED,448,sign,0.073,13663.947,0,0,6000
ED,448,verify,0.194,5163.605,0,0,9352
ML-DSA,44,key gen,0.061,16483.718,28672,1,30752
ML-DSA,44,sign,0.301,3320.623,50176,1,30288
ML-DSA,44,verify,0.079,12652.606,29696,1,29296
ML-DSA,65,key gen,0.125,7996.553,48128,1,28704
ML-DSA,65,sign,0.374,2676.720,77824,1,30288
ML-DSA,65,verify,0.106,9463.507,49152,1,29088
ML-DSA,87,key gen,0.165,6065.983,80896,1,28704
ML-DSA,87,sign,0.524,1907.410,120832,1,30288
ML-DSA,87,verify,0.180,5550.100,81920,1,29088
```

Raw Benchmarking CSV Data (WITH Assembly Optimizations)

```
Algorithm,key size,operation,avg ms,ops/sec,heap_bytes,heap_allocs,stack_bytes,
ML-KEM 512 ,128,key gen,0.008,127099.121,3072,1,31192
ML-KEM 512 ,128,encap,0.008,121799.303,6656,1,32216
ML-KEM 512 ,128,decap,0.011,89322.142,6656,1,32624
ML-KEM 768 ,192,key gen,0.015,65209.966,6144,1,30672
ML-KEM 768 ,192,encap,0.016,64215.671,10752,1,30864
ML-KEM 768 ,192,decap,0.019,52073.332,10752,1,32736
ML-KEM 1024,256,key gen,0.024,41208.981,10240,1,30656
ML-KEM 1024,256,encap,0.025,40153.694,15872,1,30848
ML-KEM 1024,256,decap,0.031,32540.409,15872,1,32720
ECC   [      SECP256R1],256,key gen,0.015,65742.789,0,0,13016
ECDHE [      SECP256R1],256,agree,0.055,18099.060,0,0,30288
ECDSA [      SECP256R1],256,sign,0.022,45838.788,0,0,26520
ECDSA [      SECP256R1],256,verify,0.057,17486.667,0,0,29376
ECC   [      SECP224R1],224,key gen,0.562,1780.723,15640,5,12368
ECDHE [      SECP224R1],224,agree,0.542,1844.871,31280,10,24336
ECDSA [      SECP224R1],224,sign,0.641,1560.335,15640,5,27104
ECDSA [      SECP224R1],224,verify,0.375,2666.851,56818,20,21664
ECC   [      SECP384R1],384,key gen,0.044,22924.394,0,0,13592
ECDHE [      SECP384R1],384,agree,0.175,5707.074,0,0,34576
ECDSA [      SECP384R1],384,sign,0.065,15398.447,0,0,26904
ECDSA [      SECP384R1],384,verify,0.184,5421.764,0,0,33760
ECC   [      SECP521R1],528,key gen,0.084,11934.199,0,0,14424
ECDHE [      SECP521R1],528,agree,0.310,3224.009,0,0,41080
ECDSA [      SECP521R1],528,sign,0.141,7106.134,0,0,28080
ECDSA [      SECP521R1],528,verify,0.343,2911.636,0,0,40160
CURVE,25519,key gen,0.030,33798.202,0,0,2664
CURVE,25519,agree,0.029,34237.440,0,0,3160
ED,25519,key gen,0.011,91000.176,0,0,2472
ED,25519,sign,0.012,80442.367,0,0,4984
ED,25519,verify,0.038,26050.499,0,0,5256
CURVE,448,key gen,0.162,6164.578,0,0,2472
CURVE,448,agree,0.164,6098.840,0,0,4160
ED,448,key gen,0.070,14225.328,0,0,2600
ED,448,sign,0.074,13525.647,0,0,6000
ED,448,verify,0.192,5217.898,0,0,9352
ML-DSA,44,key gen,0.025,39791.538,28672,1,33320
ML-DSA,44,sign,0.073,13779.943,50176,1,35760
ML-DSA,44,verify,0.033,30111.997,29696,1,32128
ML-DSA,65,key gen,0.044,22648.696,48128,1,31872
ML-DSA,65,sign,0.117,8512.492,77824,1,35776
ML-DSA,65,verify,0.062,16259.223,49152,1,32144
ML-DSA,87,key gen,0.071,14174.455,80896,1,31872
ML-DSA,87,sign,0.142,7022.552,120832,1,35776
ML-DSA,87,verify,0.084,11857.538,81920,1,32144
```

wolfSSL PQC Readiness and Migration Efforts

Hybrids

We support them!

- D(TLS) 1.3
  - SecP256r1MLKEM512
  - SecP384r1MLKEM768
  - SecP521r1MLKEM1024
  - ML-DSA hybridized with ECC via Chimera dual algorithm certificates as specified in the 2019 edition of the X.509 standard
  - X9.146 (Banking standards body) TLS 1.3 extensions for dual algorithm verification
- SSH
  - mlkem768nistp256-sha256 Key exchange

Open Source Project Integrations

- **Web Servers**
  - Apache
    https://github.com/wolfSSL/osp/blob/master/apache-httpd/README_post_quantum.md
  - Nginx
    https://github.com/wolfSSL/wolfssl-nginx?tab=readme-ov-file#post-quantum-algorithms
  - Lighttpd
    https://github.com/wolfSSL/osp/tree/master/lighttpd/lighttpd-1.4.50/wolfStartUp
- **Secure Tunneling**
  - Stunnel
    https://github.com/wolfSSL/osp/blob/master/stunnel/5.57/README_UNIX.md#stunnel-with-experimental-post-quantum-algorithms (Might be stale.)

What Others are Doing Together with wolfSSL

Post-Quantum Readiness / NCCoE

Crypto4A and wolfSSL Interoperability Demo

CRYPTO4A

Micro Controller

Firmware Image

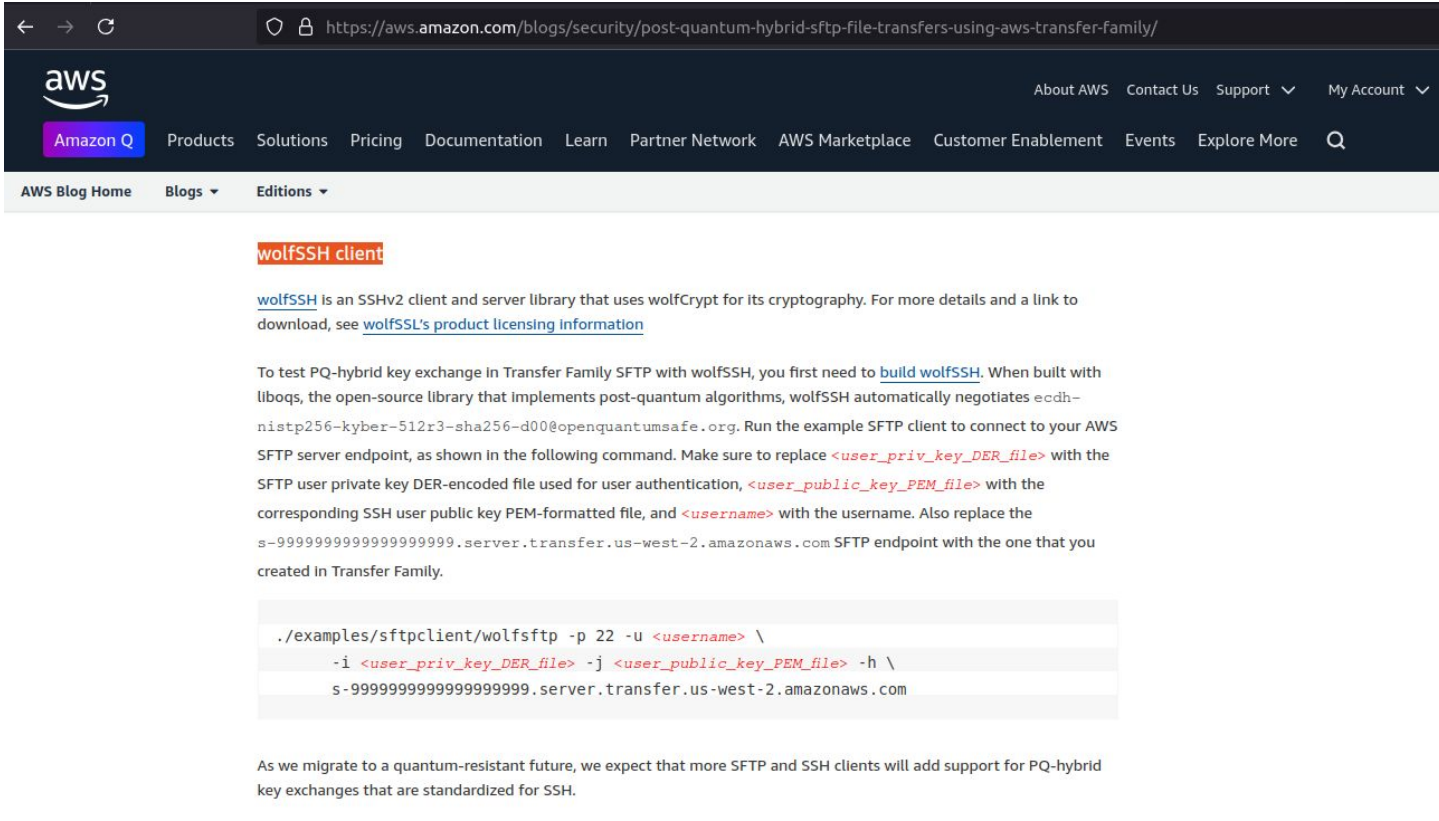
wolfboot

Firmware Signature

LMS Public Key (Stateful)

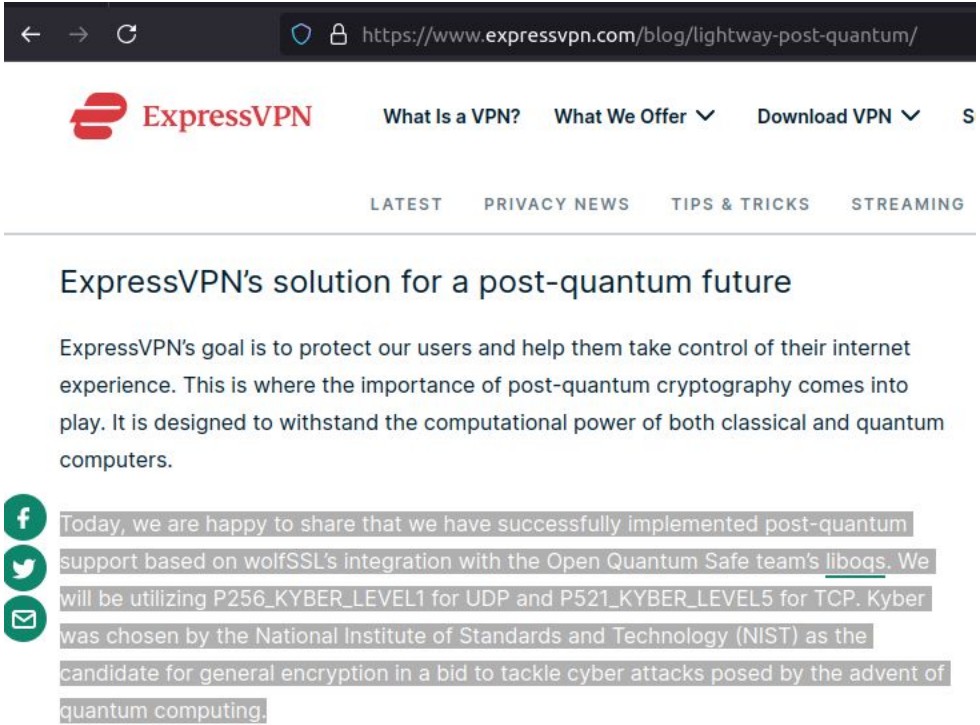
AWS Crypto and wolfSSL Interoperability Demo

https://aws.amazon.com/blogs/security/post-quantum-hybrid-sftp-file-transfers-using-aws-transfer-family/



ExpressVPN Using wolfSSL PQC in Production

https://www.expressvpn.com/blog/lightway-post-quantum/



Barriers to Migration

- **Inertia and a lack of prioritization**
- **Lack of awareness**
- **Faulty reasoning about quantum computers**
  - **The question is NOT “When will quantum computers break cryptography?”**
  - **The question is “When do I need to comply with upcoming requirements?”**

Thanks!

facts@wolfssl.com anthony@wolfssl.com