presented by

Post Quantum Cryptography impact to the UEFI Firmware

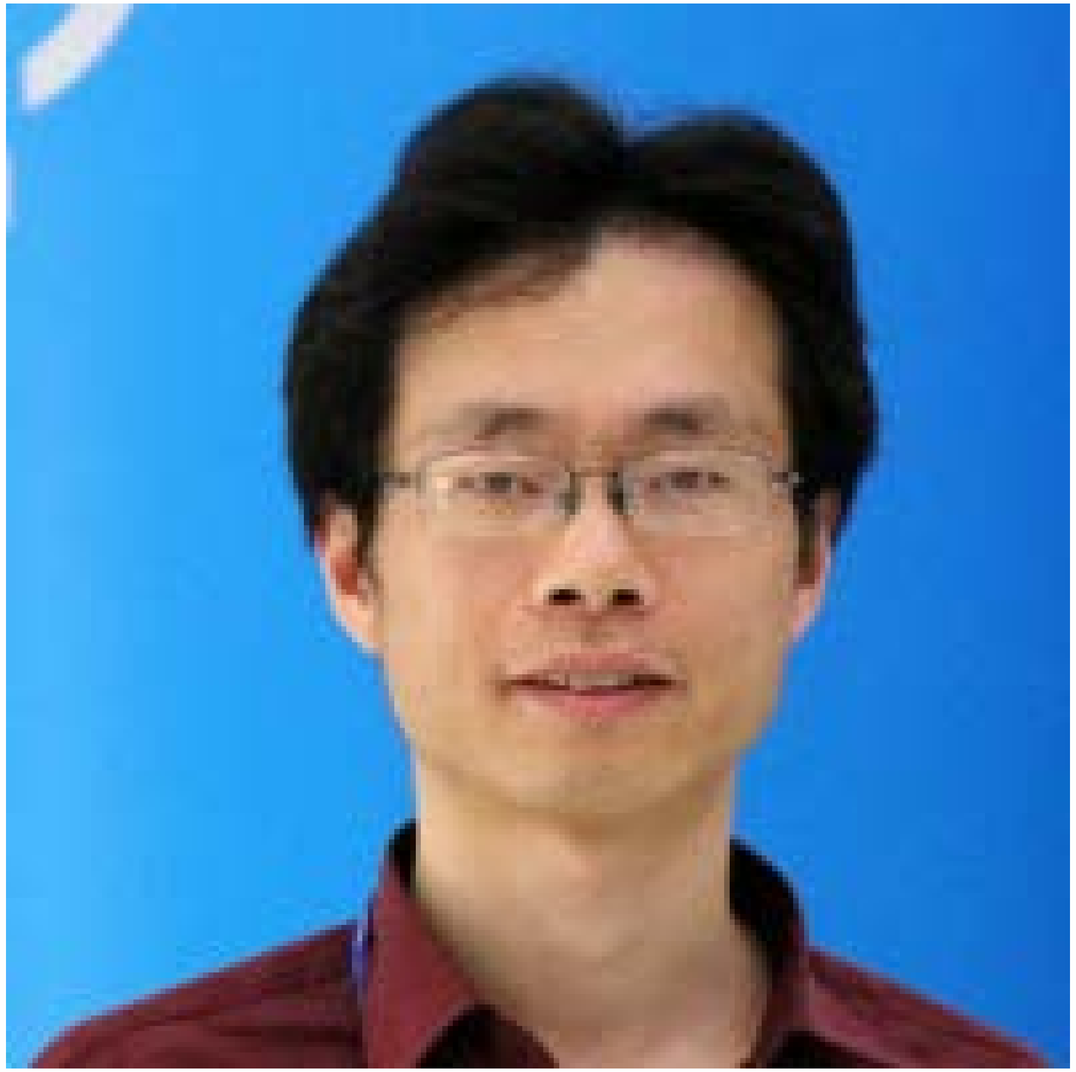
UEFI 2021 Virtual Plugfest

July 27, 2021

Jiewen Yao & Vincent Zimmer, Intel Corporation

Jiewen Yao

- **Jiewen Yao** is a principal engineer in the Intel Software and Advanced Technology Group. He has been engaged as a firmware developer for over 15 years. He is a member of the UEFI Security sub team, and the TCG PC Client sub working group.



Vincent Zimmer

- **Vincent Zimmer** is a senior principal engineer in the Intel Software and Advanced Technology Group. He has been engaged as a firmware developer for over 25 years and leads the UEFI Security sub team.

Vincent Zimmer
Intel

Agenda

- UEFI Crypto Summary
- Post Quantum Cryptography
- Enabling PQC for UEFI BIOS
- Summary / Call to Action

UEFI Crypto Summary

Cryptography in UEFI Specification

- Auth Variable

SignedData.signerInfos shall be constructed as:

- SignerInfo.version shall be set to 1.
- SignerInfo.issuerAndSerial shall be present and as in the signer's certificate.
- SignerInfo.authenticatedAttributes shall not be present.
- SignerInfo.digestEncryptionAlgorithm shall be set to the algorithm used to sign the data. Only a digest encryption algorithm of RSA with PKCS #1 v1.5 padding (RSASSA_PKCS1v1_5). is accepted.
- SignerInfo.encryptedDigest shall be present.
- SignerInfo.unauthenticatedAttributes shall not be present.

- Secure Boot

The platform key establishes a trust relationship between the platform owner and the platform firmware. The platform owner enrolls the public half of the key (PK_{pub}) into the platform firmware. The platform owner can later use the private half of the key (PK_{priv}) to change platform ownership or to enroll a Key Exchange Key. For UEFI , the recommended Platform Key format is RSA-2048. See "Enrolling The Platform Key" and "Clearing The Platform Key" for more information.

UEFI Spec Crypto Agile

- Move cryptography requirement out of UEFI specification
- https://bugzilla.tianocore.org/show_bug.cgi?id=3413

Asymmetric Cryptography in System Firmware

Usage	Category	Feature	Standard	Algorithm	Comment
Code Signing Verification	Secure Boot	**UEFI Secure Boot**	**UEFI**	**PKCS7(RSA)**	Signed one time – when the image is created.
		PI Signed FV/Section	**UEFI PI**	**PKCS7(RSA) / RSA**	
		Intel Boot Guard (Verified Boot)		RSA / SM2	
		Intel Platform Firmware Resilience (PFR)		RSA/ECDSA	
	Update	**UEFI FMP Capsule Update**	**UEFI**	**PKCS7(RSA)**	
		Intel BIOS Guard		RSA	
	Recovery	EDKII Signed Recovery with FMP Cap	**EDKII**	**RSA**	
	Report	Intel System Security Report (PPAM)		PKCS7()	
Configuration Data Signing Verification	Policy	Intel TXT Launch Control Policy (LCP)		RSA	Signed one time – when the data is created.
	Update	**UEFI Auth Variable Update**	**UEFI**	**PKCS7(RSA)**	
		Intel FSP Configuration Update		RSA	
Authentication	Device	SPDM Device Authentication	**DMTF**	**RSA/ECDSA**	Runtime Signing based upon challenge.
		SPDM Device Measurement Verification	**DMTF**	**RSA/ECDSA**	
Secure Session Establishment	Device	SPDM Session	**DMTF**	**FFDHE/ECHDE**	Key Exchange with SIGMA protocol.
	Network	HTTPS Boot (TLS)	**IETF**	**ECDHE**	

Symmetric Cryptography in System Firmware

Usage	Category	Feature	Stadard	Algorithm	Comment
Measured Boot	SRTM	TCG Trusted Boot	**TCG**	SHA2 / SM3 (TPM2.0)	SHA1 (TPM1.2) It should be deprecated
		Intel Boot Guard (Measured Boot)		SHA2 / SM3	
	DRTM	Intel Trusted Boot Technology (TXT)		SHA2 / SM3	
	Trusted VM	Intel Trust Domain Extensions (TDX)		SHA2	
Configuration Security	UEFI Variable	RPMC Variable (tbd)	EDKII	HMAC	
		RPMB Variable	**NVMe/eMMC/UFS**		
		Encrypted Variable (tbd)	EDKII	AES	
Authentication	Network	iSCSI CHAP	**IETF**	MD5	iSCSI MD5 is not allowed. Industry added SHA1/SHA2/SHA3 for ISCSI. (*) Empty means the password is send to the peer directly.
		RedFish Password	**DMTF**	-	
	Storage	HDD Password	**ATA**	-	
		OPAL Password	**TCG**	-	
	Device	SPDM Device Pre-shared Key (PSK)	**DMTF**	HMAC	
	BIOS	BIOS Setup Password	EDKII	SHA2	
Secure Session	Device	SPDM Session	**DMTF**	AEAD	ENC + MAC (TLS1.2)
	Network	HTTPS Boot (TLS)	**IETF**	AEAD (TLS1.3)	

Current Security Strength

Security Strength (Bit)	Collision Resistance (SHA)	Preimage Resistance (HMAC), HKDF	Encryption	Finite Field Crypto (DHE)	Integer Factorization Crypto (RSA)	Elliptic Curve Crypto (ECDHE, ECDSA)
112				DH-2048	RSA-2048	
128	SHA-256	SHA1	AES-128	DH-3072	RSA-3072	ECC-256
192	SHA-384			DH-7680	RSA-7680	ECC-384
256	SHA-512	SHA-256	AES-256	DH-15360	RSA-15360	ECC-521

- CNSA Suite guidance from NSA
  - SHA-384, AES-256, DH-3072, RSA-3072, ECC-384

Challenge – Quantum Computing

- Shor’s Algorithm
  - Break asymmetric algorithms (RSA, DH, ECC)
  - Break them by resolving hard-problem (factoring, discrete-log, elliptic curve)

- Grover’s Algorithm
  - Reduce security of symmetric algorithm (AES, SHA)
  - Reduce the security length to half, by brute force search.

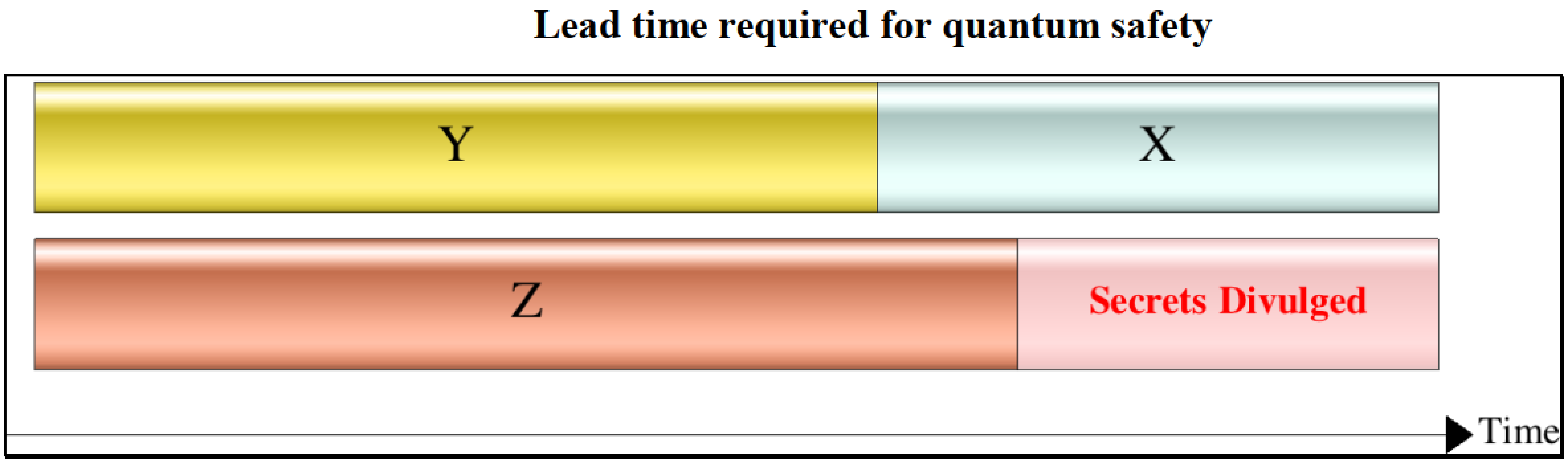
Security Strength With Quantum

Security Strength (Bit)	Collision Resistance (SHA)	Preimage Resistance (HMAC), HKDF	Encryption	Finite Field Crypto (DHE)	Integer Factorization Crypto (RSA)	Elliptic Curve Crypto (ECDHE, ECDSA)
0				DH-*	RSA-*	ECC-*
64	SHA-256		AES-128			
128	SHA-512	SHA-256	AES-256			

What is the replacement for asymmetric crypto algorithm?

Why Its Important

- **Mosca's Theorem**
  - x: "how many years we need our encryption to be secure"
  - y: "how many years it will take us to make our IT infrastructure quantum-safe“
  - z: "how many years before a large-scale quantum computer will be built“
  - If X+Y>Z, then we have a problem now, and immediate action needs to be taken

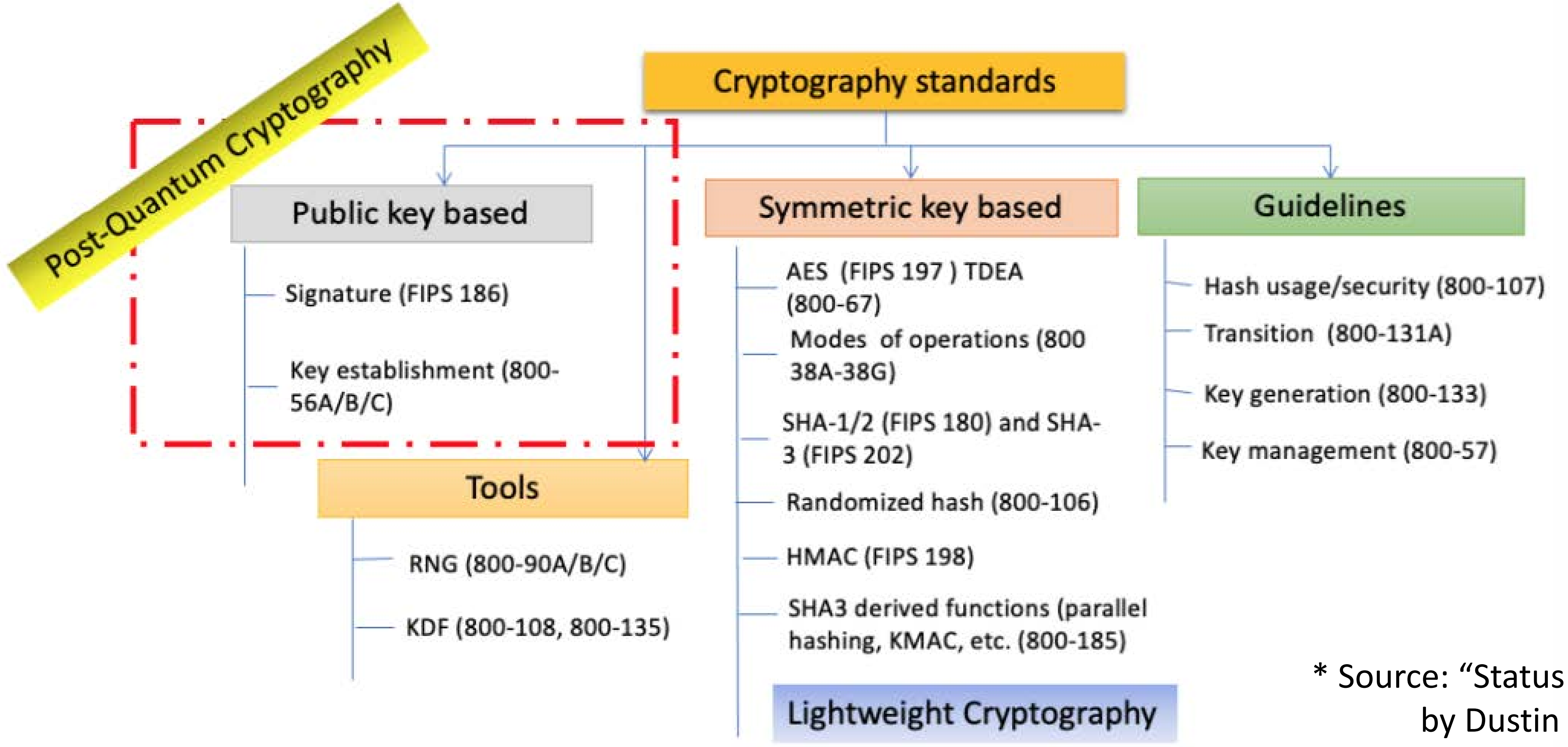


* Source: "Quantum Safe Cryptography and Security: An Introduction, Benefits, Enablers and Challenges", ETSI Whitepaper, 2015

Post Quantum Cryptography

NIST Post Quantum Cryptography

- Project was anonced at 2016
- **Goal**: develop cryptographic systems that are secure against both quantum and classical computers, and can interoperate with existing communications protocols and networks.



* Source: “Status Update on the 3rd Round” by Dustin Moody - NIST, 2021

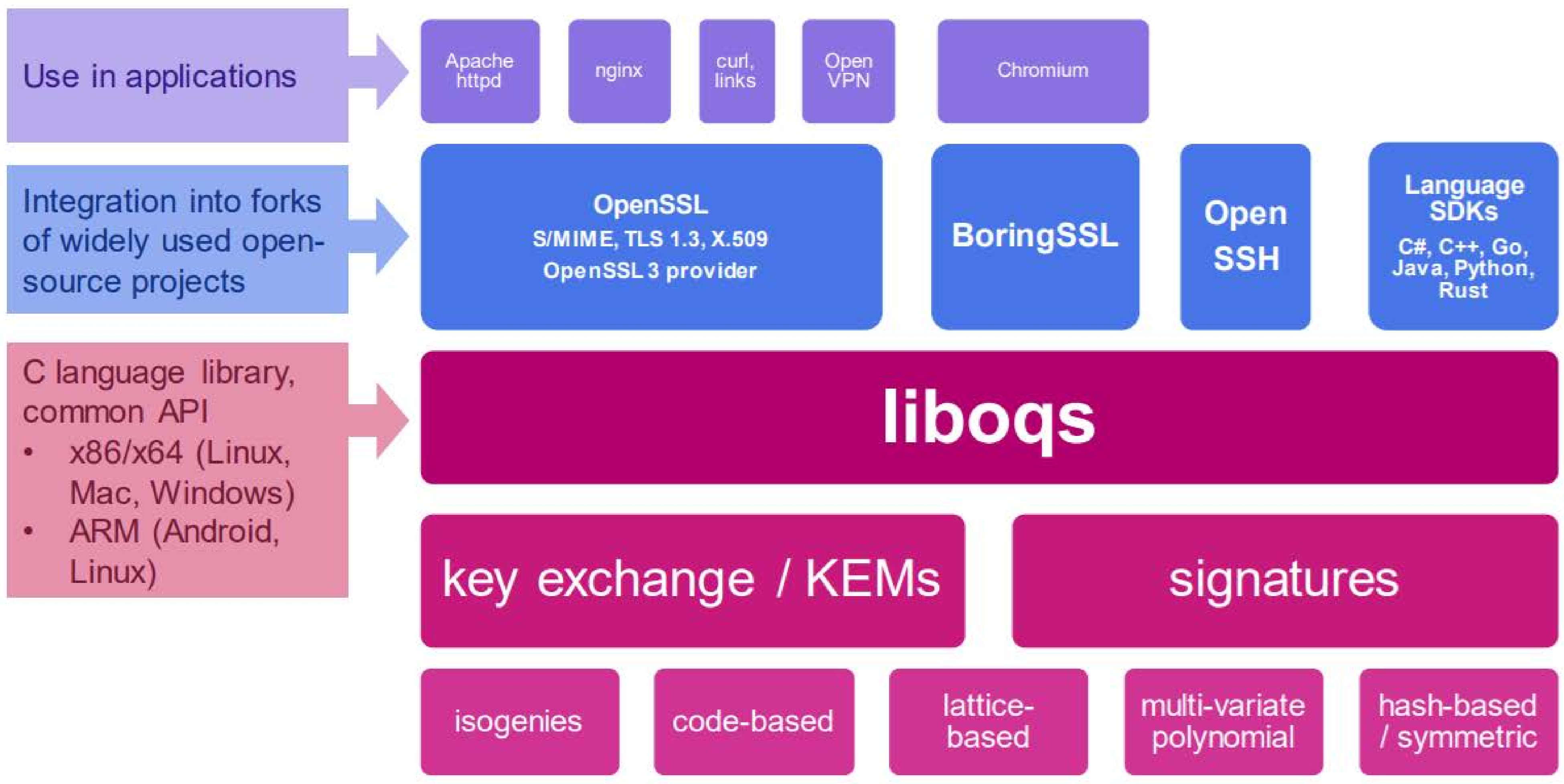
NIST Post Quantum Cryptography

- **Current Status**: Round-3
  - **Public-key Encryption** and **Key-establishment** Algorithms (4 Finalist + 5 Alternative)
  - **Digital Signature** Algorithms (3 Finalist + 3 Alternative)
- **Plan**: Release draft and call for public comment (2022~2023)
- **Summary**

Usage	Algorithm	Hard Problem
Public-key Encryption and Key-establishment	**Classic McEliece**, **BIKE**, **HQC**	Code
	Kyber, **NTRU**, **SABER**, **FrodoKEM**, **NTRUprime**	Lattice
	SIKE	Isogeny
Digital Signature	**SPHINCS+**, **Picnic**	Symmetric (Hash)
	Dilithium, **Falcon**	Lattice
	Rainbow, **GeMSS**	Multivariate

Open Quantum Safe (OQS) Project

- **Goal**: Support the development and prototyping of quantum-resistant cryptography.
  - https://openquantumsafe.org/, https://github.com/open-quantum-safe



* Source: "Updates from the Open Quantum Safe project" by John Schanck - University of Waterloo, 2021

Open Quantum Safe (OQS) Project

- **Current Status**: liboqs
  - MIT license
  - Implementations from PQClean or direct contribution
  - **C language** with wrapper to go, java, .net, python, rust.
  - Can be integrated to boringssl, openssh, openssl.
  - **Release:** 0.5.0, 0.6.0
- **Algorithm (0.6.0)**
  - **Key Establishment:** bike, classic_mceliece, frodokem, hqc, kyber, ntru, ntruprime, saber, sike.
  - **Digital Signature:** dilithium, falcon, picnic, rainbow, sphincs.
  - match NIST candidate round 3 (only miss gemss signature)
- **UEFI POC integration**
  - To be discussed later …

PQC_KEM (PubKey and CipherText size)

Algorithm	Parameter	Public Key Size (Bytes)	Secret Key Size (Bytes)	Cipher Text Size (Bytes)	Shared Secret Size
BIKE	BIKE-L{1,3}	2542~6206	3110~13236	2542~6206	32
Classic-McEliece	{348864, 460896, 6688128, 6960119, 8192128}	261120~1357824	6452~14080	128~240	32
FrodoKEM	FrodoKEM-{640,976,344}	9616~21520	19888~43088	9720~21632	16~32
HQC	HQC-{128,192,256}	2249~7245	2289~7285	4481~14469	64
Kyber	Kyber-{512,768,1024}	800~1568	1632~3168	768~1568	32
NTRU	HPS-{2048-509,2048-677,4096-821}, HRSS-701	699~1138	935~1450	699~1138	32
NTRUprime	ntrulpr{653,761,857}	897~1322	1125~1999	897~1312	32
Saber	{LightSaber,Saber,FireSaber}	672~1312	1568~3040	736~1472	32
SIKE	SIDH-p{434,503,610,751} SIKE-p{434,503,610,751}	197~564 197~564	28~48 350~640	197~564 236~596	110-188 16~32

PQC_SIG (PubKey and Sig size)

Algorithm	Parameter	Public Key Size (Bytes)	Secret Key Size (Bytes)	Signature Size (Bytes)
Dilithium	Dilithium{2,3,5}	1312~2592	2528~4864	2420~4595
Falcon	Falcon-{512,1024}	897~1793	1281~2305	690~1330
Picnic	picnic_L{1,3,5} picnic3_L{1,3,5}	33~65 35~65	49~97 52~97	34036~209510 14612~61028
Rainbow	Rainbow-{I,III,V} Rainbow-{I,III,V}-{compress}	60192~1930600 60192~1930600	103648~1408736 64	66~212 66~212
SPHINCS+	SPHINCS+-{SHA,SHAKE}-{128,192,256}	32~64	64~128	8080~49216

Transition Plan – Hybrid Mode

- **Hybrid Mode** (NIST SP800-56C, NIST.CSWP.04282021)
  - you can combine an unapproved (i.e. a PQC) algorithm with a NIST-approved algorithm and still receive FIPS validation

- For example:
  - hybrid (PQC_KEM + ECDHE) key exchange in TLS 1.3
  - hybrid (PQC_SIG + RSA/ECDSA) authentication in TLS 1.3
  - Hybrid (PQC_SIG + RSA/ECDSA) X.509 certificate

NIST Stateful Hash-Based Signature

- **Published**: NIST SP800-208 - Recommendation for **Stateful Hash-Based Signature** Schemes, 2020
  - RFC8391 - **XMSS**: eXtended Merkle Signature Scheme
  - RFC8554 - Leighton-Micali Hash-Based Signatures (**LMS**)
  - **Limited Usage**: *An application that may fit this profile is **the authentication of firmware updates** for constrained devices*.

Usage	Algorithm	Parameter (both RFC + NIST)	RFC only	NIST only
Digital Signature	LMS	LMOTS_{SHA256}_N{32}_W{1,2,4,8}	-	Hash=SHAKE, N=24
		LMS_{SHA256}_M{32}_H{5,10,15,20,25}	-	Hash=SHAKE, M=24
	XMSS XMSS^MT	WOTPS_{SHA2}_{256}	Hash=SHAKE, n=512	Hash=SHAKE, n=192
		XMSS_{SHA2}_{10,16,20}_{256} XMSS^MT_{SHA2}_{20/{2,4},40/{2,4,8},60/{3,6,12}}_{256}	Hash=SHAKE, n=512	Hash=SHAKE, n=192

NIST Stateful Hash-Based Signature

- HBS can only sign **a limited number** of messages

Alg	Param	Signature Number (2^H)	Sign Size (bytes)	PubKey Size (Bytes)
LMS (HSS)	H10	2^10 = 1 K	1456	76
	H15	2^15 = 32 K	1616	76
	H25	2^25 = 32 M	1936	76
	H15/H10	2^(15 + 10) = 32 M	3172	76
	H25/H15	2^(25 + 15) = 1 T	3652	76
XMSS/ XMSS^MT	h=10	2^10 = 1 K	2500	68
	h=16	2^16 = 64 K	2692	68
	h=20	2^20 = 1 M	2820	68
	h=20, d = 2	2^20 = 1 M	4963	68
	h=40, d = 4	2^40 = 1 T	9893	68

Hash Based Signature - reference

- **LMS**
  - https://github.com/cisco/hash-sigs
    - C language, BSD3 license, RFC8554: draft-mcgrew-hash-sigs-07.
  - https://github.com/davidmcgrew/hash-sigs
    - Python, BSD3 license, RFC8554: draft-mcgrew-hash-sigs-05.
- **XMSS**
  - https://github.com/XMSS/xmss-reference
    - C language, CC0 1.0 license, RFC8391
  - https://github.com/mkannwischer/xmssfs
    - C language, RFC8391 forward secure implementation (based upon xmss-reference)
  - https://github.com/lothar1998/XMSS-tree
    - pyhton, MIT license, RFC8391
  - https://github.com/openssh/openssh-portable
    - Integrate XMSS to SSH

- **UEFI POC integration**
  - To be discussed later …

Enabling PQC for UEFI BIOS

Potential PQC usage in UEFI

- General Asymmetric Cryptography usage (round 3)
  - **Key Establishment** (TLS, SPDM session)
  - **Digital Signature** (Runtime Challenge/Response)
    - *Need wait for NIST PQC announcement*
    - *TLS – UEFI, SPDM – Maybe in PEI/SMM*

- Special Usage
  - **Stateful Hash Based Signature** (LMS, XMSS)
  - Secure Boot, Capsule Update, Signed Recovery
    - Secure Boot – UEFI
    - Signed Update – DXE/SMM
    - Signed Recovery - PEI

liboqs in EDKII

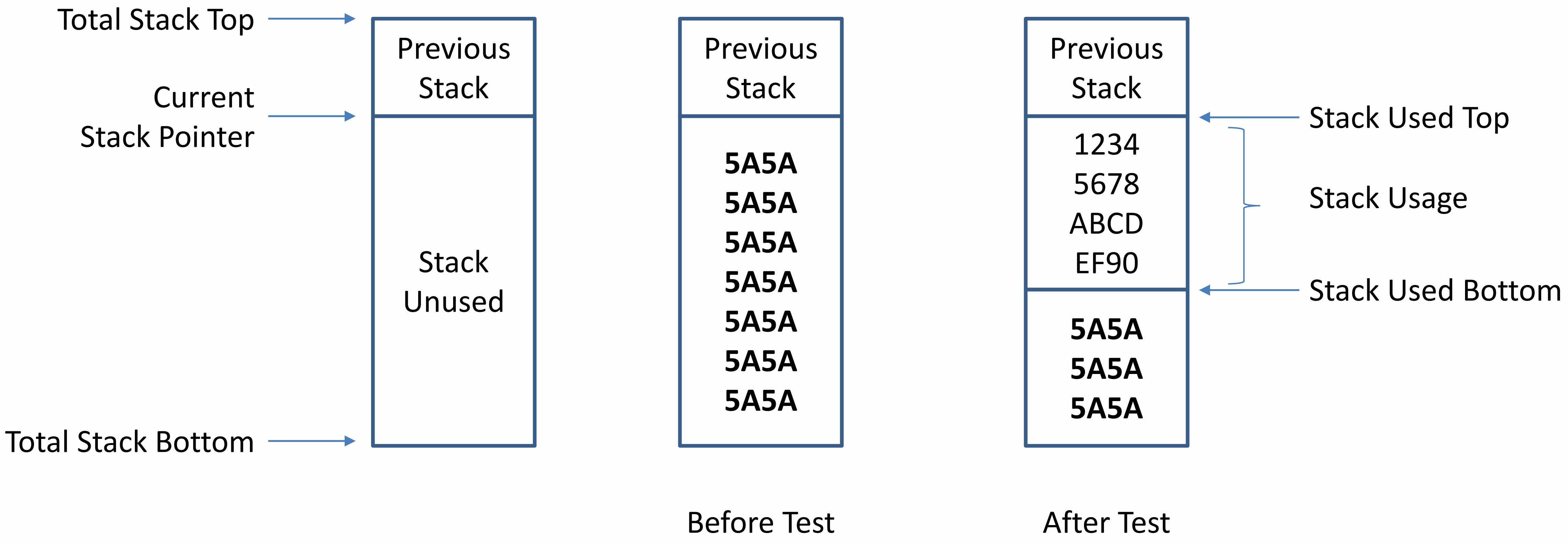
- Advantage:
  - **Common Interface** : OQS_SIG_new (AlgName), OQS_KEM_new (AlgName)
  - **Traditional Crypto Dependency** (AES, SHA2, SHA3, RAND) : Self-contained (or) openssl
  - **Arch Specific Acceleration**: (X86: SSE/AVX, ARM: SHA2/AES)
  - **No Global Variable** (Context in Stack or Heap)
- Challenge:
  - **Build:** liboqs uses CMAKE, EDKII uses INF.
  - **Compiler dependency**
    - PQClean Algorithms are OK
    - BIKE only works with GCC.
    - HQC cause _chkstk link error with MSVC.
  - **Special CPU instruction not enabled** (yet)
    - AVX/AVX2 (CLASSIC_MCELIECE, HQC, KYBER, NTRU, NTRUPRIME, SABER, DILITHIUM, FALCON, SPHINCS+)
  - **Large Stack usage** (up to 4M) – *to be discussed later…*

Stack/Heap limitation in UEFI

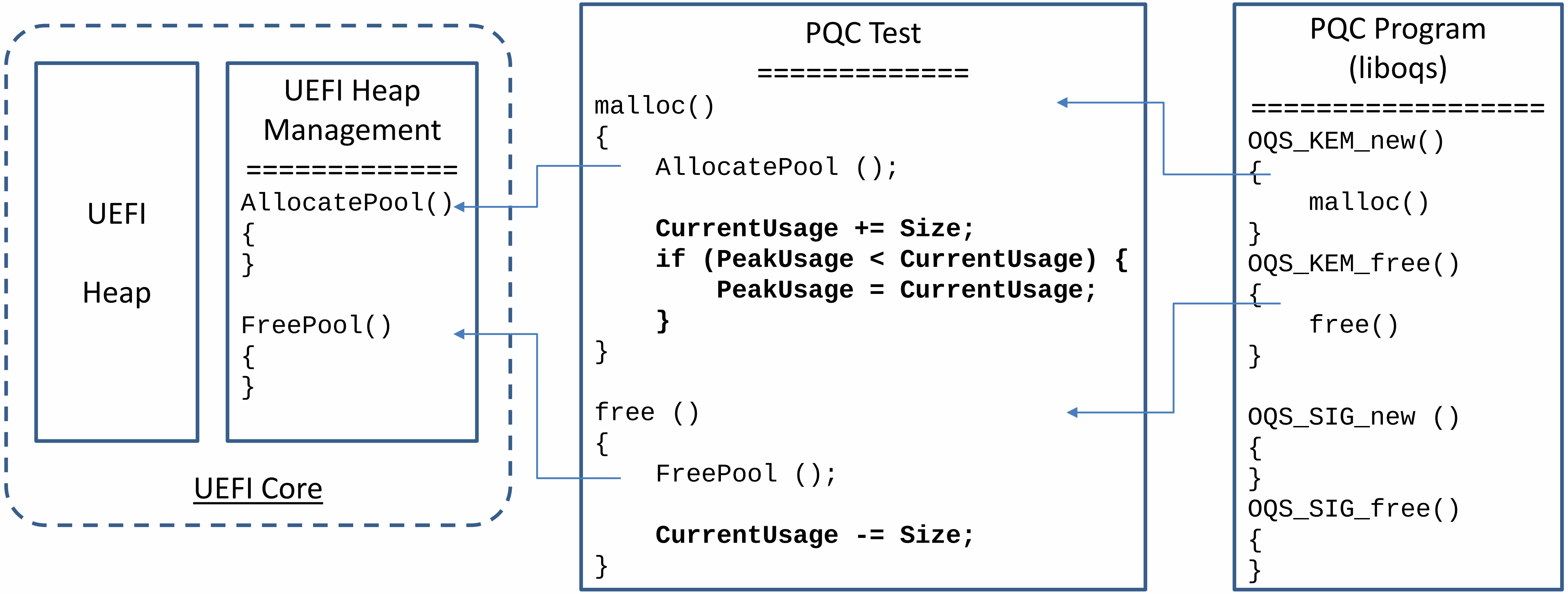
- typical memory size in each phase

Env	Stack	Heap
UEFI	UEFI: 128K as minimal STACK_SIZE = **128K**	Physical Memory Size
PEI (PreMem)	Heap / 2 = PeiStackSize (Or) PcdPeiTemporaryRamStackSize	Cache As Ram (CAR) size: PcdOvmfSecPeiTempRamSize (**64K**)
PEI (PostMem)	Heap / 2 = NewStackSize (Or) PcdPeiCoreMaxPeiStackSize (**128K**)	PEI_MIN_MEMORY_SIZE (**320M**) S3: mS3AcpiReservedMemorySize (**512K**)
SMM	PcdCpuSmmStackSize = **8K** (default)	SMRAM Size (**8M**)

How to: Stack Usage Calculation



How to: Heap Usage Calculation



UEFI liboqs (kem) – Stack/Heap

- **liboqs** key establishment memory usage: (KeyGeneration + shared key calculation)
- MSVC, X64 build.

Algorithm	Parameter	Stack (KB)	Heap (KB)
Classic-McEliece	Classic-McEliece-348864 Classic-McEliece-{460896,6688128,8192128}	2153 4657	262 526~1341
Kyber	Kyber-{512,768,1024}	11~23	4~7
NTRU	NTRU	26~41	3~5
NTRUprime	NTRUprime	13~20	4~5
Saber	{LightSaber, Saber, FireSaber}	11~22	4~7
FrodoKEM	FrodoKEM-{AES,Shake}	79~213	39~85
SIKE	{SIKE,SIDH} {SIKE,SIDH}-compressed	7~13 68~188	2 2

UEFI liboqs (sig) – Stack/Heap

- **liboqs** digital signature memory usage: (KeyGeneration + Signing + Verification)
- MSVC, X64 build.

Algorithm	Parameter	Stack (KB)	Heap (KB)
Dilithium	Dilithium{2,3,5}	52~123	7~12
Falcon	Falcon-{512,1024}	43~83	4~6
picnic	picnic_L{1,3,5} picnic3_L{1,3,5}	7 5	173~906 1398~5964
Rainbow	Rainbow-I Rainbow-III Rainbow-V	175~318 971~1726 2143~3774	60~260 260~1474 525~3262
SPHINCS+	SPHINCS+-{SHA,SHAKE}-{128,192,256}	4~9	9~49

HBS (lms hash-sigs, xmss-reference) in EDKII

- Common Attribute:
  - **Only for limited use cases**: Secure Boot, Capsule Update, Signed Recovery
  - **Only verification is required. (Don't GenKey in UEFI, very slow)**
- Challenge:
  - **Build:** Makefile v.s. INF in EDKII.
  - **API inconsistent**
    - **xmss-reference** veries message directly - XMSS_SIGN_OPEN()
    - **hash-sigs** does not assume fit all messages into memory.
      - hss_validate_signature_init()/hss_validate_signature_update()/hss_validate_signature_finalize().
  - **Compiler dependency**
    - **xmss-reference** uses variable length array (VLA). Need change to MAX size for MSVC build.
  - **Execution env dependency**
    - **Xmss-reference** hardcodes random from "/dev/urandom". (not issue for verification)

UEFI HBS (lms hash-sigs, xmss-reference) – Stack/Heap

- **HBS** memory usage: (Verification only)
  - LMOTS_SHA256_N32_W8, LMS_SHA256_M32_H?
  - WOTPS_SHA2_256, XMSS_SHA2_?_256, XMSS^MT_SHA2_?_256

Algorithm	Parameter	Stack (KB)	Heap (KB)
LMS	H5, H10, H15	5	2
	H10/H10, H15/H10, H15/H15	5	4
XMSS XMM^MT	h:10,16,20	20	8~9
	h/d: 20/2, 40/4, 60/6	20	14~44

Other Data

- **Prototype**
  - Available at https://github.com/jyao1/CryptoEx
  - Support:
    - liboqs (PQC SIG/KEM) integration
    - hash-sigs (LMS) integration
    - xmss-reference (XMSS) integration

- **Performance Data**
  - Refer to "Post-Quantum LMS and SPHINCS+ Hash-Based Signatures for UEFI Secure Boot"
  - *"None of the proposed parameter sets perform verification slower than **7ms**, which is satisfactory. "*
  - See below:

Parameter	Keys (B)		Verifier (KB)		Keygen (s)	Sign (Mcycles)		Verify (Mcycles)	
	Priv	Pub	Code	Stack		Mean	Stdv	Mean	Stdv
LMS256H15W4	48	60	2.57	1.81	2.519	1.145	0.051	0.370	0.033
LMS256H15W8	48	60	2.15	1.81	13.720	6.237	0.302	2.855	0.290
LMS256H20W4	48	60	2.57	1.81	3.222	1.465	0.037	0.373	0.026
LMS256H20W8	48	60	2.15	1.81	19.373	8.807	0.555	2.857	0.274

Additional Challenge

- **PKCS7** signed data (PE, Capsule, AuthVar)
  - Need integrate PQC algorithm to PKCS7.
  - (or) Use raw signature data. (e.g. FV, Section)

- **X509** certificate
  - Public key size + Signature size
  - May bigger than 64K
- **Key Exchange** data
  - Public key size + Exchange Ciphertext size
  - May bigger than 64K

UEFI/PI Data Structure

- UEFI Variable (Window Requirement)
  - PcdMaxAuthVariableSize – individual AuthVar, **64K**
    - Storage the signature database - certificate
  - PcdFlashNvStorageVariableSize – total var storage, **128K at least**
    - OVMF: **256K**
  - Variable data can be from Hob – GetHobVariableStore().
- Hob
  - HobLength – **UINT16** (**64K**)
  - Need special handling.
- FFS File
  - File Size – **UINT8[3]** (**16M**)
  - Need use FFS Header2 ExtendedSize – UINT64
- File Section
  - Section Size – **UINT8[3]** (**16M**)
  - Need use SectionHeader2 ExtendedSize – UINT32

Beyond UEFI – TLS protocol

- TLS (Transport Layer Security) protocol
  - Usage in firmware : **HTTPS boot**.
  - TLS include: Public Certificate, Signature, KeyExchange Data (PublicKey or CipherText)
  - Refer to "Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH"

- Prototype
  - Avalaible at https://github.com/open-quantum-safe/openssl
  - with https://github.com/open-quantum-safe/liboqs
  - Support **hybrid mode**.

Beyond UEFI – SPDM protocol

- DMTF SPDM (Secure Protocol and Data Model) protocol
  - Usage in firmware: **Device Authentication/Measurement, session communication**.
  - SPDM include: Public Certificate, Signature, KeyExchange Data (PublicKey or CipherText)
  - GetCertificate() Length/Offset: use **UINT16 length** – **64K** at most.
  - SPDM Secure Message – use **UINT16 length** – **64K** at most.
- PCI Data Object Exchange (DOE)
  - SPDM over DOE
  - Transport Length: use 18bit for DWORD – **1M** at most.

- Prototype
  - Available at https://github.com/jyao1/openspdm-pqc
  - Based upon liboqs and openssl-oqs.
  - Define **PQC algorithm**. Support **hybrid mode**.
  - Enhance spdm to allow it transport large packet.

Beyond UEFI – TPM

- Future TPM (https://futuretpm.eu/)
  - Post-Quantum Cryptography TPM
  - Limitation:
    - IO Buffer Size: **4096 bytes** default.
    - Computation time: XMSS takes long time to gen keys.
    - NVRam size limitation: XMSS keys/state.
    - Internal Cache: need store XMSS cache data for optimization.

- Prototype
  - PQC TPM and TSS

Summary & Call for Action

Summary & Call for Action

- The industry is preparing post-quantum cryptography (PQC).

- We should prepare for PQC and consider crypto agile design with hybrid mode.
  - Feedback to https://bugzilla.tianocore.org/show_bug.cgi?id=3413

- We should consider the PQC implementation in resource constrain environment.

Reference – Whitepaper / Guide

- NIST Post Quantum Cryptography - https://csrc.nist.gov/projects/post-quantum-cryptography
- NIST Presentation: NIST Status Update on the 3rd Round - https://csrc.nist.gov/CSRC/media/Presentations/status-update-on-the-3rd-round/images-media/session-1-moody-nist-round-3-update.pdf
- NIST Presentation: Update From the Open Quantum Safe Project - https://csrc.nist.gov/CSRC/media/Presentations/updates-from-the-open-quantum-safe-project/images-media/session-9-schanck-update-open-safe-project.pdf
- NIST Whitepaper - Getting Ready for Post-Quantum Cryptography - https://csrc.nist.gov/publications/detail/white-paper/2021/04/28/getting-ready-for-post-quantum-cryptography/final
- M. Mosca, “Setting the Scene for the ETSI Quantum-safe Cryptography Workshop”, 2013, http://docbox.etsi.org/Workshop/2013/201309_CRYPTO/e-proceedings_Crypto_2013.pdf
- ETSI whitepaper, “Quantum Safe Cryptography and Security: An Introduction, Benefits, Enablers and Challenges,” 2015, https://portal.etsi.org/Portals/0/TBpages/QSC/Docs/Quantum_Safe_Whitepaper_1_0_0.pdf

- NIST SP800-208 - Recommendation for Stateful Hash-Based Signature Schemes - https://csrc.nist.gov/publications/detail/sp/800-208/final
- NIST SP800-57 Part 1 - Recommendation for Key Management: Part 1 – General - https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final
- NIST SP800-56C - Recommendation for Key-Derivation Methods in Key-Establishment Schemes - https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final
- RFC8554 - XMSS: eXtended Merkle Signature Scheme - https://www.rfc-editor.org/rfc/rfc8554.txt
- RFC8391 - Leighton-Micali Hash-Based Signatures - https://www.rfc-editor.org/rfc/rfc8391.txt

Reference – Document (Paper)

- Quantum Algorithm Zoo - https://quantumalgorithmzoo.org/
- Peter W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science,* 1994, https://pdfs.semanticscholar.org/6902/cb196ec032852ff31cc178ca822a5f67b2f2.pdf
- Peter. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," in SIAM Journal of Computing 1997, https://arxiv.org/pdf/quant-ph/9508027
- John Proos, Christof Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," in *Quantum Information & Computation 2003*, https://arxiv.org/pdf/quant-ph/0301141.pdf
- Martin Roetteler, Michael Naehrig, Krysta M. Svore, Kristin Lauter, "Quantum resource estimates for computing elliptic curve discrete logarithms," in *ASIACRYPT 2017*, https://arxiv.org/pdf/1706.06752
- Lov K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings 28th Annual ACM Symposium on the Theory of Computing,* 1996, https://arxiv.org/pdf/quant-ph/9605043.pdf

- Daniel J. Bernstein, Tanja Lange, "Post-quantum cryptography - dealing with the fallout of physics success", 2017, https://eprint.iacr.org/2017/314
- Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri, "NIST Post-Quantum Cryptography - A Hardware Evaluation Study", 2019, https://eprint.iacr.org/2019/047
- Eric Crockett, Christian Paquin, and Douglas Stebila "Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH", 2019, https://eprint.iacr.org/2019/858
- Panos Kampanakisy Dimitrios Sikeridis, "Two Post-Quantum Signature Use-cases - Non-issues, Challenges and Potential Solutions", 2020, https://eprint.iacr.org/2019/1276
- Dimitrios Sikeridis, Panos Kampanakis, Michael Devetsikiotis, "Post-Quantum Authentication in TLS 1.3 - A Performance Study", 2020, https://eprint.iacr.org/2020/071.pdf
- PANOS KAMPANAKIS, PETER PANBURANA, MICHAEL CURCIO, CHIRAG SHROFF, MD MAHBUB ALAM, "Post-Quantum LMS and SPHINCS+ Hash-Based Signatures for UEFI Secure Boot", 2021, https://eprint.iacr.org/2021/041.pdf

Reference - Project

- Post-Quantum Cryptography
  - liboqs, https://github.com/open-quantum-safe/liboqs
  - Openssl-oqs, https://github.com/open-quantum-safe/openssl
  - pqclean, https://github.com/PQClean
  - libpqcrypto, https://libpqcrypto.org/

- Hash Based Signature
  - LMS, https://github.com/cisco/hash-sigs
  - XMSS, https://github.com/XMSS/xmss-reference
  - XMSS with forward Secure, https://github.com/mkannwischer/xmssfs
  - SSH with XMSS, https://github.com/openssh/openssh-portable

- UEFI/Firmware Prorotype
  - https://github.com/jyao1/CryptoEx
  - https://github.com/jyao1/openspdm-pqc

Questions?

Thanks for attending the UEFI 2021 Virtual Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by