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.
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 \((PK_{\text{pub}})\) into the platform firmware. The platform owner can later use the private half of the key \((PK_{\text{priv}})\) 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](https://bugzilla.tianocore.org/show_bug.cgi?id=3413)
# Asymmetric Cryptography in System Firmware

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

<table>
<thead>
<tr>
<th>Usage</th>
<th>Category</th>
<th>Feature</th>
<th>Standard</th>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured Boot</td>
<td>SRTM</td>
<td>TCG Trusted Boot</td>
<td>TCG</td>
<td>SHA2 / SM3 (TPM2.0)</td>
<td>SHA1 (TPM1.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intel Boot Guard (Measured Boot)</td>
<td></td>
<td>SHA2 / SM3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRTM</td>
<td>Intel Trusted Boot Technology (TXT)</td>
<td></td>
<td>SHA2 / SM3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trusted VM</td>
<td>Intel Trust Domain Extensions (TDX)</td>
<td></td>
<td>SHA2</td>
<td></td>
</tr>
<tr>
<td>Configuration Security</td>
<td>UEFI Variable</td>
<td>RPMC Variable (tbd)</td>
<td>EDKII</td>
<td>HMAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RPMB Variable</td>
<td>NVMe/eMMC/UFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted Variable (tbd)</td>
<td>EDKII</td>
<td>AES</td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td>Network</td>
<td>iSCSI CHAP</td>
<td>IETF</td>
<td>MD5</td>
<td>iSCSI MD5 is not allowed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RedFish Password</td>
<td>DMTF</td>
<td>-</td>
<td>Industry added SHA1/S HA2/S HA3 for iSCSI. (*)</td>
</tr>
<tr>
<td></td>
<td>Storage</td>
<td>HDD Password</td>
<td>ATA</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPAL Password</td>
<td>TCG</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Device</td>
<td>SPDM Device Pre-shared Key (PSK)</td>
<td>DMTF</td>
<td>HMAC</td>
<td>Empty means the password is send to the peer directly.</td>
</tr>
<tr>
<td></td>
<td>BIOS</td>
<td>BIOS Setup Password</td>
<td>EDKII</td>
<td>SHA2</td>
<td></td>
</tr>
<tr>
<td>Secure Session</td>
<td>Device</td>
<td>SPDM Session</td>
<td>DMTF</td>
<td>AEAD</td>
<td>ENC + MAC (TLS1.2)</td>
</tr>
<tr>
<td></td>
<td>Network</td>
<td>HTTPS Boot (TLS)</td>
<td>IETF</td>
<td>AEAD (TLS1.3)</td>
<td></td>
</tr>
</tbody>
</table>
## Current Security Strength

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

- **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) Comparison

<table>
<thead>
<tr>
<th>Security Strength (Bit)</th>
<th>Collision Resistance (SHA)</th>
<th>Preimage Resistance (HMAC), HKDF</th>
<th>Encryption</th>
<th>Finite Field Crypto (DHE)</th>
<th>Integer Factorization Crypto (RSA)</th>
<th>Elliptic Curve Crypto (ECDHE, ECDSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>DH-*</td>
<td>RSA-*</td>
<td>ECC-*</td>
</tr>
<tr>
<td>64</td>
<td>SHA-256</td>
<td>AES-128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>SHA-512</td>
<td>SHA-256</td>
<td>AES-256</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

Post Quantum Cryptography
NIST Post Quantum Cryptography

- **Project** was announced 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**

<table>
<thead>
<tr>
<th>Usage</th>
<th>Algorithm</th>
<th>Hard Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public-key Encryption and Key-establishment</td>
<td>Classic McEliece, BIKE, HQC</td>
<td>Code</td>
</tr>
<tr>
<td></td>
<td>Kyber, NTRU, SABER, FrodoKEM, NTRUprime</td>
<td>Lattice</td>
</tr>
<tr>
<td></td>
<td>SIKE</td>
<td>Isogeny</td>
</tr>
<tr>
<td>Digital Signature</td>
<td>SPHINCS+, Picnic</td>
<td>Symmetric (Hash)</td>
</tr>
<tr>
<td></td>
<td>Dilithium, Falcon</td>
<td>Lattice</td>
</tr>
<tr>
<td></td>
<td>Rainbow, GeMSS</td>
<td>Multivariate</td>
</tr>
</tbody>
</table>
Open Quantum Safe (OQS) Project

- **Goal:** Support the development and prototyping of quantum-resistant cryptography.

*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](#), [openssl](#), [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)

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

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

- **Hybrid Mode** ([NIST SP800-56C](https://www.nist.gov/) [NIST.CSWP.04282021](https://www.nist.gov/))
  - 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](https://www.nist.gov/publications/recommendation-stateful-hash-based-signature-schemes) - Recommendation for Stateful Hash-Based Signature Schemes, 2020
  - [RFC8391](https://tools.ietf.org/html/rfc8391) - **XMSS:** eXtended Merkle Signature Scheme
  - [RFC8554](https://tools.ietf.org/html/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.

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

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

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

• **LMS**
  - [https://github.com/cisco/hash-sigs](https://github.com/cisco/hash-sigs)
    - C language, BSD3 license, RFC8554: draft-mcgrew-hash-sigs-07.
  - [https://github.com/davidmcgrew/hash-sigs](https://github.com/davidmcgrew/hash-sigs)
    - Python, BSD3 license, RFC8554: draft-mcgrew-hash-sigs-05.

• **XMSS**
  - [https://github.com/XMSS/xmss-reference](https://github.com/XMSS/xmss-reference)
    - C language, CC0 1.0 license, RFC8391
  - [https://github.com/mkannwischer/xmssfs](https://github.com/mkannwischer/xmssfs)
    - C language, RFC8391 forward secure implementation (based upon xmss-reference)
    - Python, MIT license, RFC8391
  - [https://github.com/openssh/openssh-portable](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**

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

www.uefi.org
# How to: Stack Usage Calculation

<table>
<thead>
<tr>
<th>Total Stack Top</th>
<th>Previous Stack</th>
<th>Previous Stack</th>
<th>Previous Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Stack Pointer</td>
<td>Stack Unused</td>
<td>5A5A</td>
<td>1234</td>
</tr>
<tr>
<td>Total Stack Bottom</td>
<td></td>
<td>5A5A</td>
<td>5A5A</td>
</tr>
</tbody>
</table>

## Before Test
- Stack Used Top: 
  - 5A5A
- Stack Used Bottom: 
  - 5A5A

## After Test
- Stack Used Top: 
  - 1234
  - 5678
  - ABCD
  - EF90
- Stack Used Bottom: 
  - 5A5A
  - 5A5A
  - 5A5A
How to: Heap Usage Calculation

UEFI Heap Management

AllocatePool()
{
}
FreePool()
{
}

PQC Test

malloc()
{
    AllocatePool();
    CurrentUsage += Size;
    if (PeakUsage < CurrentUsage) {
        PeakUsage = CurrentUsage;
    }
}
free()
{
    FreePool();
    CurrentUsage -= Size;
}

PQC Program (liboqs)

OQS_KEM_new()
{
    malloc()
}
OQS_KEM_free()
{
    free()
}
OQS_SIG_new ()
{
}
OQS_SIG_free()
UEFI liboqs (kem) – Stack/Heap

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

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

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

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Stack (KB)</th>
<th>Heap (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>H5, H10, H15</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>H10/H10, H15/H10, H15/H15</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>XMSS</td>
<td>h:10,16,20</td>
<td>20</td>
<td>8~9</td>
</tr>
<tr>
<td>XMM^MT</td>
<td>h/d: 20/2, 40/4, 60/6</td>
<td>20</td>
<td>14~44</td>
</tr>
</tbody>
</table>
Other Data

- **Prototype**
  - Available at [https://github.com/jyao1/CryptoEx](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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Keys (B)</th>
<th>Verifier (KB)</th>
<th>Keygen (s)</th>
<th>Sign (Mcycles)</th>
<th>Verify (Mcycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Priv</td>
<td>Pub</td>
<td>Code</td>
<td>Stack</td>
<td>Mean Stdv</td>
</tr>
<tr>
<td>LMS256H15W4</td>
<td>48</td>
<td>60</td>
<td>2.57</td>
<td>1.81</td>
<td>2.519 1.145 0.051</td>
</tr>
<tr>
<td>LMS256H15W8</td>
<td>48</td>
<td>60</td>
<td>2.15</td>
<td>1.81</td>
<td>13.720 6.237 0.302</td>
</tr>
<tr>
<td>LMS256H20W4</td>
<td>48</td>
<td>60</td>
<td>2.57</td>
<td>1.81</td>
<td>3.222 1.465 0.037</td>
</tr>
<tr>
<td>LMS256H20W8</td>
<td>48</td>
<td>60</td>
<td>2.15</td>
<td>1.81</td>
<td>19.373 8.807 0.555</td>
</tr>
</tbody>
</table>
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
  – Available 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](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

Reference – Document (Paper)

• Quantum Algorithm Zoo - https://quantumalgorithmzoo.org/


Reference - Project

- **Post-Quantum Cryptography**
  - liboqs, [https://github.com/open-quantum-safe/liboqs](https://github.com/open-quantum-safe/liboqs)
  - pqclean, [https://github.com/PQClean](https://github.com/PQClean)
  - libpqcrypto, [https://libpqcrypto.org/](https://libpqcrypto.org/)

- **Hash Based Signature**
  - LMS, [https://github.com/cisco/hash-sigs](https://github.com/cisco/hash-sigs)
  - XMSS with forward Secure, [https://github.com/mkannwischer/xmssfs](https://github.com/mkannwischer/xmssfs)
  - SSH with XMSS, [https://github.com/openssh/openssh-portable](https://github.com/openssh/openssh-portable)

- **UEFI/Firmware Prorotype**
  - [https://github.com/jyao1/CryptoEx](https://github.com/jyao1/CryptoEx)
  - [https://github.com/jyao1/openspdm-pqc](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