Quantum Resistance and Confidential Computing

John Manferdelli
johnmanferdelli@Hotmail.com
June 11, 2024
"I could talk about crypto endlessly --- this is not a threat."

Historical cryptographic cipher suite algorithms

- Public key
  - RSA-1024, then RSA-2048 then (2002) ECC-256, ECC-384 then ECC-512
  - Used for authentication and “key encapsulation”
- Symmetric Key
  - DES(56-bit keys) then (circa 2000) AES-128, AES-192, AES-256
  - For good symmetric cipher with k-bit key, security is thought to be 2^k
  - “Bulk encryption”
- Cryptographic hashes
  - MD-4 (128), then MD5(128) then SHA-1(160), then SHA-2 (256, 384, 512), then SHA-3 (variable length)
  - Strength of cryptographic hash with k-bit output is thought to be $2^{k/2}$
  - Message digest, mixing, digital currency

CC Provides a Foundation for Security

Four capabilities of a Confidential Computing:

- **Isolation.** Program address space and computation.
- **Measurement.** Use cryptographic hash to create an unforgeable program identity.
- **Secrets.** Isolated storage and exclusive program access. (aka, "sealed storage").
- **Attestation.** Enable remote verification of program integrity and secure communication with other such programs.

CC provides principled security wherever your programs run and wherever your data resides *even if you don't operate the computers the programs run on.*

Isolation and measurement Hash

- Program address space isolated
- Program hashed to give non-forgeable identity

Secrets

- Seal: protect a secret for this measurement
- Unseal: restore a secret for this measurement

Symmetric cipher

Attestation

- Statement signed by a trusted party (HW) that specifies
  - Program identity (measurement) program
  - Hardware protection (isolation, integrity, confidentiality) guarantees
  - Statement attributable to isolated entity

Public key

Hard Problems and Public Key Systems

- RSA : Integer factorization
- ECC : Discrete log in an Elliptic curve
- Newer:
  - Shortest vector in a lattice
    - Learning with Error
    - Ring Learning with Error

Rationalizing crypto suites (2002)

- Idea is to make all the algorithms in a crypto suite “equally” difficult to break
  - Example: AES-128 is equivalent to SHA-256
  - Example: AES-256 is equivalent to SHA-512

ECC	RSA	AES
163	1024	
256	3072	128
384	7680	192
521	15360	256

Government standards pre-2017 (“Suite B”)

- “Secret” level
  - Public key
    - RSA-2048
    - ECC-256
  - Hash
    - SHA-2 (256)
  - Symmetric
    - AES-128

- “Top Secret” level
  - Public key
    - RSA-3072
    - ECC-521
  - Hash
    - SHA-2 (512)
  - Symmetric
    - AES-256

Government standards 2020 (Suite-B)

- No more levels
- Public key
  - RSA-2048
  - ECC-384
- Hash
  - SHA-2 (384)
- Symmetric
  - AES-256

Government standards 2024 (CNSA 2.0)

- Public key
  - RSA-3072
  - ECC-384
- Hash
  - SHA-2 (384)
- Symmetric
  - AES-256

Enter Peter Shor

- Quantum Computers
  - Qubits, superposition, entanglement and all that (spooky action at a distance)
  - Physical vs Logical Qubits
- Shor’s algorithm (1994)
  - Uses Quantum Fourier Transform
  - “Breaking” integer factorization and discrete log takes about the same asymptotic time as public encryption --- oops
- But
  - No quantum computer in sight
  - National Academy of Sciences, “Quantum Computing: Progress and Prospects” (2018)

NIST Competition

NIST: "We better get some public key algorithms that are rooted in quantum safe hard problems"

- Most candidates are based on "shortest vector in a (high dimensional) lattice" (SVP)
  - Nice, provable properties about "worst case hardness" vs "average hardness" and chosen ciphertext adaptive resistance
  - Two classes of implementations
    - Learning with errors (Solving simultaneous equations over a finite field with errors")
      - Frodo
    - Ring learning with errors (Same but over a ring like $\mathbb{Z}_p[x]/(x^n + 1)$)
      - NTRU, Crystal

The winner is

- FIPS-203, FIPS-204 (April 2024)
- Crystal-Dilithium
  - For signing/authentication
- Crystal-Kyber
  - For key encapsulation

Both based on the R-LWE hard problem, but they are different algorithms

Lattice hard problems have some nice properties (Piekert, Regev)

Lattices

- The set $\Lambda = \mathbb{Z}b_1 + \mathbb{Z}b_2 + \ldots + \mathbb{Z}b_n$, where $b_1, b_2, \ldots, b_n$ are linearly independent is called a lattice.
- $\Lambda^* = \{y \in \mathbb{Z}^n : (x, y) \in \mathbb{Z}, \forall x \in \Lambda\}$
- $vol(\Lambda) = \det(b_1, b_2, \ldots, b_n)$, where $b_1, b_2, \ldots, b_n$ are the generators of Λ. Note that any set of generators will do since they are related by uni-modular transformations.
- Let Λ be a lattice
  - The CVP problem is: Find $v \in \Lambda$: $||v|| = min_{w \in \Lambda, w \neq 0}(||w||)$
  - The CVP_γ problem is: Find $v \in \Lambda$: $||v|| \leq \gamma \cdot min_{w \in \Lambda, w \neq 0}(||w||)$

LWE

- Based on solving noisy linear equations $mod\ q$. Choose $\overrightarrow{a_i} \in \mathbb{Z}_q{}^n$ uniformly at random. $\vec{s} \in \mathbb{Z}_q{}^n$ is a secret and $m \geq n$ approximate equations $\overrightarrow{a_i} \cdot \vec{s} = b_i\ (mod\ q)$. Errors, $e_1, e_2, \dots, e_n$ are chosen from distribution χ. Reduces to LWE. Chris Peikert et. al.
- Search LWE problem: Given the above, find $\vec{s}$.
- Decision LWE: Distinguish with non-negligible probability, between $\vec{b} = A\vec{s}$+ $\vec{e}$ and $\vec{b} \in \mathbb{Z}_q{}^m$ chosen uniformly at random given $A, \vec{b}$
- Peikert's results show it is possible to pick parameters so that solving the cipher is equivalent to solving worst-case LWE

LWE cryptosystem

- Given $(n \geq m, l, t, r, q, \chi)$ where χ is a probability distribution $\mathbb{Z}_q$, message space is ${\mathbb{Z}_2}^l$ and cipher space is ${\mathbb{Z}_q}^n \times {\mathbb{Z}_q}^l$.
- Key Gen
  1. Choose $S \in {\mathbb{Z}_q}^{n\times l}$, uniformly from the distribution χ.
  2. Choose $A \in {\mathbb{Z}_q}^{m\times n}$, and $E \in {\mathbb{Z}_q}^{m\times l}$ uniformly from the distribution χ.
  3. Private key is S, public key is $(A, P = AS + E)$
- Encrypt
  1. For $\vec{v} \in {\mathbb{Z}_2}^l$, choose $\vec{a} \in \{0,1\}^m$, uniformly at random
  2. $\overrightarrow{CT} = (\vec{u} = A^T\vec{a}, \vec{c} = P^T\vec{a} + \upharpoonleft \frac{q}{2} \downharpoonleft \vec{v}))$
- Decrypt
  1. Compute $\upharpoonleft (\upharpoonleft \frac{q}{2} \downharpoonleft)^{-1}(\vec{c} - S^T\vec{u})$ ⇃ $(mod\ 2)$

- Decryption may have errors. Suppose χ is a discrete Gaussian $D_{\mathbb{Z},s}$. Then $E^T\vec{a}$ has magnitude $\leq \sqrt{ms}$ with high probability. Error occurs if $E^T\vec{a} \geq \frac{q}{4}$. One can show that for any n, $\exists q, m, s$ such that the error is small and the underlying LWE problem is hard.

LWE example

- $n = 4, q = 23, m = 8, \alpha = \frac{5}{23}, s = 5, \sigma = \frac{s}{\sqrt{2\pi}}$

- $A = \begin{bmatrix} 9 & 5 & 11 & 13 \\ 13 & 6 & 6 & 2 \\ 6 & 21 & 17 & 18 \\ 22 & 19 & 20 & 8 \\ 2 & 17 & 10 & 21 \\ 10 & 8 & 17 & 11 \\ 5 & 16 & 12 & 2 \\ 5 & 7 & 11 & 7 \end{bmatrix}, S = \begin{bmatrix} 5 & 2 & 9 & 1 \\ 6 & 8 & 19 & 1 \\ 19 & 18 & 9 & 18 \\ 9 & 2 & 14 & 18 \end{bmatrix}$

From Heiko Knopse

LWE example

- $$E = \begin{bmatrix} 0 & 22 & 1 & 21 \\ 0 & 22 & 22 & 22 \\ 6 & 21 & 17 & 18 \\ 22 & 22 & 22 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 1 & 22 & 1 & 22 \\ 22 & 0 & 0 & 1 \end{bmatrix}, \mathrm{P} = \begin{bmatrix} 10 & 5 & 21 & 7 \\ 3 & 1 & 13 & 1 \\ 19 & 15 & 6 & 13 \\ 22 & 22 & 22 & 0 \\ 9 & 20 & 20 & 17 \\ 15 & 21 & 1 & 2 \\ 0 & 12 & 3 & 19 \\ 16 & 2 & 7 & 15 \end{bmatrix},$$

From Heiko Knopse

LWE example

- Encrypt $m = (1,0,1,1)^T$, using a $= (1,1,0,1,0,0,0,1)^T$
  - $\lfloor \frac{23}{2} m \rceil = (12,0,12,12)^T$,
  - $(u, c) = \left(A^T a, P^T a + \lfloor \frac{23}{2} m \rceil\right) = ((3,14,2,7)^T, (4,5,7,5)^T)(mod\ 23)$

- Decrypt:
  - $m' = c - S^T u = (11,21,12,10)^T\ (mod\ 23)$,
  - $\lfloor \frac{1}{12} m' \rceil\ (mod\ 2) = (1,0,1,1)^T$

From Heiko Knopse

Ring-LWE

- Put $R = \frac{\mathbb{Z}_q[x]}{x^n+1}, n = 2^k, R \approx {\mathbb{Z}_q}^n$. $a \in R$, generates ideal (a) corresponding to a q-ary ideal lattice.

- Ring LWE: Given a $\in R$, and $b = as + e$, for $\mathrm{s}, \mathrm{e} \in R$, find s.

- Solving R-LWE is at least as hard as solving CVP_γ on arbitrary ideal lattices

Some common features of Dilithium and Kyber

- Ring is $\mathbb{Z}_p[x]/(x^n + 1)$ in both cases
  - $p = 3329$ for Kyber
  - $p = 2^{23} - 2^{13} + 1$ for Dilithium
  - So. the same modular arithmetic we all grew up with
- $n = 256$ in both cases and there is a primitive 512th root of unity for both primes (You are not expected to understand this).
  - As a result, $x^n + 1$ factors into 128 quadratics
  - Allows us to perform a "Number Theory Transform" that turns convolution into pointwise multiplication for ring operations giving a nice speedup

Dilithium (simplified)

- Remember $A^{k\times l}$ is generated randomly from $R = \mathbb{Z}_p[x]/(x^{256}+1)$.
- s_1 is a vector of dimension l with entries from R has random coefficients $\leq \eta$
- s_2 is a vector of dimension k with entries from R has random coefficients $\leq \eta$
- $t = As_1 + s_2$

Sign

$y \coloneqq {S_{\gamma_1 - 1}}^l$
$w_1 \coloneqq \mathrm{highbits}(\mathrm{Ay}, 2\gamma_2)$
$c \coloneqq SH(M||w_1)$
$z \coloneqq y + cs_1$
return(z, c)

Verify

${w_1}' \coloneqq \mathrm{highbits}(\mathrm{Az} - \mathrm{ct}, 2\gamma_2)$
$c' \coloneqq SH(M||{w_1}')$
Check $c' == c$ AND $||z||_\infty < \gamma_1 - \beta$

Dilithium

Parameters: $(p = 8380417, R = \frac{\mathbb{Z}_p}{x^{256}+1}, k = 5, l = 4, \gamma_1 = \frac{p-1}{16}, \gamma_2 = \frac{\gamma_1}{2}, \eta = 5, \beta = 275)$

- KeyGen
  - $A \in R^{k \times l}$, selected from random distribution over R
  - $(s_1, s_2) \in {S_\eta}^k \times {S_\eta}^l$, selected at random, ${S_\eta}^k$ consists of elements of R^k with coefficients $\leq \eta$
  - Set $t = As_1 + s_2$
  - Public key is (A, t), Private key is (s_1, s_2)

  For the sake of compression A is generated from a seed and SHAKE-256

Dilithium

- Sign(pk, sk, M) --- simplified
  1. $z = \perp$
  2. while ($z = \perp$) {
  3. $\quad y = S_{\gamma_1}{}^{l} - 1$
  4. $\quad w_1 = highbits(Ay, 2\gamma_2)$
  5. $\quad c = SHAKE - 256(M||w_1)$
  6. $\quad z = y + cs_2$
  7. $\quad$ if ($||z||_\infty \geq \gamma_1 - \beta$) OR $lowbits(Ay - cs_2, 2\gamma_1) \geq \gamma_2 - \beta$) then $z = \perp$
  8. }

  Signature is (z, c)

Dilithium

- Verify(pk, M,z, c) --- simplified
  1. $w_1' = highbits(Az - ct, 2\gamma_2)$
  2. Return true if $||z||_\infty \leq \gamma_1 - \beta$ AND $c = SHAKE - 256(M||w_1')$, otherwise return false

Kyber

- Parameters: $(p = 3329, R = \frac{\mathbb{Z}_p}{x^{256}+1}, k = 4, \eta = 2), \hat{x} = NTT(x)$
- KeyGen
  1. $\hat{t} = \hat{A}\hat{s} + \hat{e}$, A is generated from seed ρ
- Encrypt(m, r) [$r \in R^k$ is generated from CDB_{η_1} , e_1 is generated from CDB_{η_2}
  1. $u(x) = NTT^{-1}(\hat{A}^T) + e_1$
  2. $\mu = decompress_1(decode_1(m)), v = NTT^{-1}(\hat{t}^T \cdot r + e_1 + \mu)$
  3. $c_1 = encode_{d_u}(compress_{d_u}(u)), c_2 = encode_{d_v}\left(compress_{d_v}(r)\right)$
  4. Return (c_1, c_2)
- Decrypt(c_1, c_2)
  1. $w = v - NTT^{-1}(\hat{s} \cdot NTT(u))$
  2. Return $encode_1(compress_1(w))$

Kyber

- Parameters: $(p = 3329, R = \frac{\mathbb{Z}_p}{x^{256}+1}, k = 4, \eta = 2), \hat{x} = NTT(x)$

- EKM-KeyGen
  1. z is a random 32-byte value
  2. $(e_{PKE}, d_{PKE}) = KeyGen$
  3. $d_{KEM} = d_{PKE}||H(e_{PKE})||z$
  4. Return (e_{PKE}, d_{KEM})

Kyber

- Parameters: $(p = 3329, R = \frac{\mathbb{Z}_p}{x^{256}+1}, k = 4, \eta = 2)$, $\hat{x} = NTT(x)$
- EKM-Encaps
  1. m is a random 32-byte value
  2. $(K, r) = SHA - 3_{512}(m||H(e_h))$
  3. $c = kyber - encrypt(ek, m, r)$
  4. Return (K, c)
- EKM-Decaps
  1. $m' = kyber - decrypt(dk, c)$
  2. $(K', r') = SHA - 3_{512}(m'||H(e_k))$
  3. $\overline{K} = SHAKE - 256(z||c, 32)$
  4. $c' = kyber - encrypt(e_k, m', r')$
  5. If $(c == c')$ return K′ else error

CONFIDENTIAL COMPUTING CONSORTIUM

Kyber

- Notes

Effect on CCC

- Current compute and key sizes
  - ”Old SGX” : ECC-256
  - AMD-SEV : RSA-4096 (512 bytes)
- New compute and key sizes
  - Compute similar
  - Key size about 1.5 KB
- Software
  - Only change in Certifier Framework for “default” use case: new suite declaration
  - Open SSL handles new algorithms so if you manage your own keys, use the new OpenSSL support
- Hardware
  - Need to implement new algorithms in HW
  - Standard instructions, reasonable storage and compute requirements

Thank you!

John Manferdelli <johnmanferdelli@hotmail.com>