(2)

CRYSTALS-Kyber

Pei He

CRYSTALS-Kyber: Introduction

- Shor's Algorithm:
  - Efficient polynomial-time algorithm using quantum computers:
    - Solve DLP and ECDLP
    - Find prime factors (RSA)
- Lattice problems such as LWE are believed to be hard for quantum computers to solve
- Kyber is based on the hard Module Learning with Errors Problem

	n	q	k	η_1	η_2	d_u	d_v	required RBG strength (bits)
ML-KEM-512	256	3329	2	3	2	10	4	128
ML-KEM-768	256	3329	3	2	2	10	4	192
ML-KEM-1024	256	3329	4	2	2	11	5	256

NIST National Institute of Standards and Technology U.S. Department of Commerce

Learning With Errors:

Given t = As + e, with e being small, find a good s

PUBLIC KEY

A | t

$$\begin{array}{rcrcrcrcl}
77x & + & 7y & + & 28z & + & 23w & = & 2859 \\
21x & + & 19y & + & 30z & + & 48w & = & 3508 \\
4x & + & 24y & + & 33z & + & 38w & = & 3848 \\
8x & + & 20y & + & 84z & + & 61w & = & 6225 \\
6x & + & 53y & + & 1z & + & 86w & = & 4886 \\
42x & + & 86y & + & 31z & + & 8w & = & 9062 \\
5x & + & 24y & + & 79z & + & 27w & = & 6103 \\
16x & + & 7y & + & 35z & + & 21w & = & 2589
\end{array}$$

e

$$\begin{array}{cr}
+ & -3 \\
+ & 2 \\
+ & -1 \\
+ & 0 \\
+ & 4 \\
+ & -1 \\
+ & -2 \\
+ & 2
\end{array}$$

PRIVATE KEY

s

$$\begin{array}{c}
10 \\
82 \\
50 \\
5
\end{array}$$

Module LWE (MLWE)

- Basically LWE except A, s, t and e go from being integer vectors to polynomial vectors
- All operations done within a polynomial ring eg. $R_q = \mathbb{Z}_{541}[x]/(x^4 + 1)$
- Polynomial coefficients can be encoded as bytes

$$A = \begin{bmatrix} 442 + 502x + 513x^2 + 15x^3 & 368 + 166x + 37x^2 + 135x^3 \\ 479 + 532x + 116x^2 + 41x^3 & 12 + 139x + 385x^2 + 409x^3 \\ 29 + 394x + 503x^2 + 389x^3 & 9 + 499x + 92x^2 + 254x^3 \end{bmatrix}$$

$$s = \begin{bmatrix} 2 - 2x + x^3 \\ 3 - 2x - 2x^2 - 2x^3 \end{bmatrix}$$

$$e = \begin{bmatrix} 2 - 2x - x^2 \\ 1 + 2x + 2x^2 + x^3 \\ -2 - x^2 - 2x^3 \end{bmatrix}$$

$$t = As + e = \begin{bmatrix} 30 + 252x + 401x^2 + 332x^3 \\ 247 + 350x + 259x^2 + 485x^3 \\ 534 + 234x + 137x^2 + 443x^3 \end{bmatrix}$$

Kyber - Key Encapsulation Mechanism

Alice

Keygen

- Randomly select **n** - length bitstream **rho** to expand into A
- Randomly select small **[-2,2]** polynomial vectors **s**, e
- **t** = A**s** + e

- Randomly select n- length bitstream z

Decapsulate

- u = decompress(u*)
- v = decompress(v*)
- **m’** = round_**q**(v’ - **s** . u’)

- **K’**, R’ = G(**m’**, H(**rho** + **t**))
- **K*** = J(z + c)
- Encrypt m’ with **rho**, **t** and R’ to get u’, v’.
- If u==u’ and v==v’ return **K’** else **K***

Default parameters:
q = 3329 and **n** = 256

Public key: **rho**, **t**

Ciphertext: **u***, **v***

Fujisaki-Okamoto transform:
using hash functions

- G: sha-512
- H: sha-256
- J: sha-256

to make KEM resistant against chosen ciphertext attack

Bob

Encapsulate

- Randomly select **n**-length bitstream **m** as secret message
- h = H(**rho** + **t**)
- **K**, R = G(**m**, h)

Encrypt

- Expand rho into A
- Select small **[-2,2]** polynomial vectors r, e1, e2 using seed R
- u = Ar + e1
- v = t . r + e2 + [q/2] m
- u* = compress(u)
- v* = compress(v)

Kyber-512 Demo

- Source code available at https://github.com/wph12/kyber

Attack - Bad RNG

Encapsulate

- Randomly select **n**-length bitstream **m** as secret message
- h = H(**rho** + **t**)
- **K**, R = G(**m**, h)

- Bob only uses bad RNG to generate message M. The rest of KEM remains the same

Attack - Kyberslash

- Side channel: timing leaks information about secrets (m, m’)
- Time taken by division depends on input!
- **Kyberslash1:**
  - Need to convert m’ from polynomial (int array)
  - to bytes representation for hashing with G

- **K’**, R’ = G(**m’**, H(**rho** + **t**))
- **K*** = J(z + c)
- Encrypt m’ with **rho**, **t** and R’ to get u’, v’.
- If u==u’ and v==v’ return **K’** else **K***

  - Some Kyber libraries will use this line of code (t is derived from m’):

```
t = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
```

  - If we obtain m’, we can easily calculate the shared secret key K’

Kyberslash1 baby demo

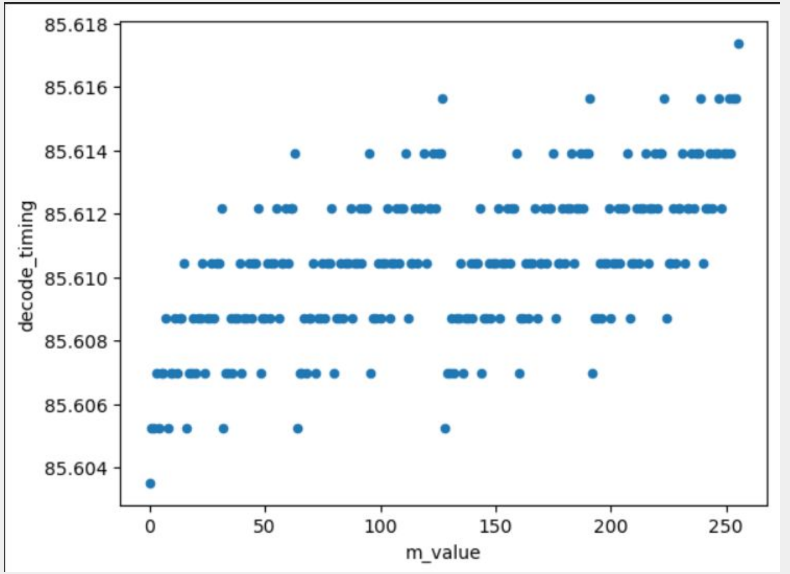
```
t  = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
```

```
score = 0
for t in m_poly:
    t += (t >> 15) & self.q
    t = ((t << 1) + self.q//2)
    score += math.log2(t)
return score
```

- Used timing score to approximate time taken
    - The higher the value of t, the longer it will take to be divided

- Results are grossly exaggerated since we assume that only the value of m will affect time taken to decode m’
    - It does show that information is leaked

- Results with 1-byte m (**n** = 8) as shown



Kyberslash2

- When compressing v, information may be leaked about m
- Again, division of secret by public parameter
- x here refers to bytes of v

```
def compress_ele(self, x, d):
    """
    Compute round((2^d / q) * x) % 2^d
    """
    t = 1 << d
    y = (t * x + 1664) // 3329  # 1664 = 3329 // 2
    return y % t
```

Encrypt

- Expand rho into A
- Select small **[-2,2]** polynomial vectors r, e1, e2 using seed R
- u = Ar + e1
- v = t . r + e2 + [q/2] m
- u* = compress(u)
- v* = compress(v)

- Other mathematical attacks
    - BKZ - lattice reduction algorithm