NTT

格子暗号への平文・鍵確認オラクルを用いた鍵回復攻撃とサイドチャネル攻撃・故障利用攻撃

草川恵太（NTT社会情報研究所）

2021/11/16–17 新世代暗号の設計・評価

Joint work with 田中裕太郎・伊東燦・上野嶺・本間尚文（東北大） 高橋順子（NTT社会情報研究所）

[XIUTH21] Xagawa, Ito, Ueno, Takahashi, Homma (ASIACRYPT 2021) https://eprint.iacr.org/2021/840
[UXTITH21] Ueno, Xagawa, Tanaka, Ito, Takahashi, Homma (TCEHS Vol. 2022, Issue 1) https://eprint.iacr.org/2021/849

Agenda

- Why PQC?
- NIST PQC
- PKE and KEM
- Key-recovery attack using oracles
- Fault-injection analysis and Side-channel analysis

Why PQC

Post-Quantum Cryptography

NTT

Quantum Computers:

Google: 54 qubits (2019)

IBM Q53: 53 qubits (2019)

USTC: 76 qubits (2020)

Why PQC?

QC solves factoring/DL [Sho94]

→  may be quantum

Countermeasure:

1. Use quantum cryptography
   QKD etc.
2. Use PQC
3. (Use looooooooooong RSA/DL)

cf. https://en.wikipedia.org/wiki/List_of_quantum_processors
[Sho94] Shor (FOCS 1994)

Post-Quantum RSA?

Why PQC?

QC solves factoring/DL [Sho94]

→  may be quantum

… RSA is broken?

Post-Quantum RSA [BFHLV17]:

pqrsa30: 2^{30}-byte keys using 1024-bit primes

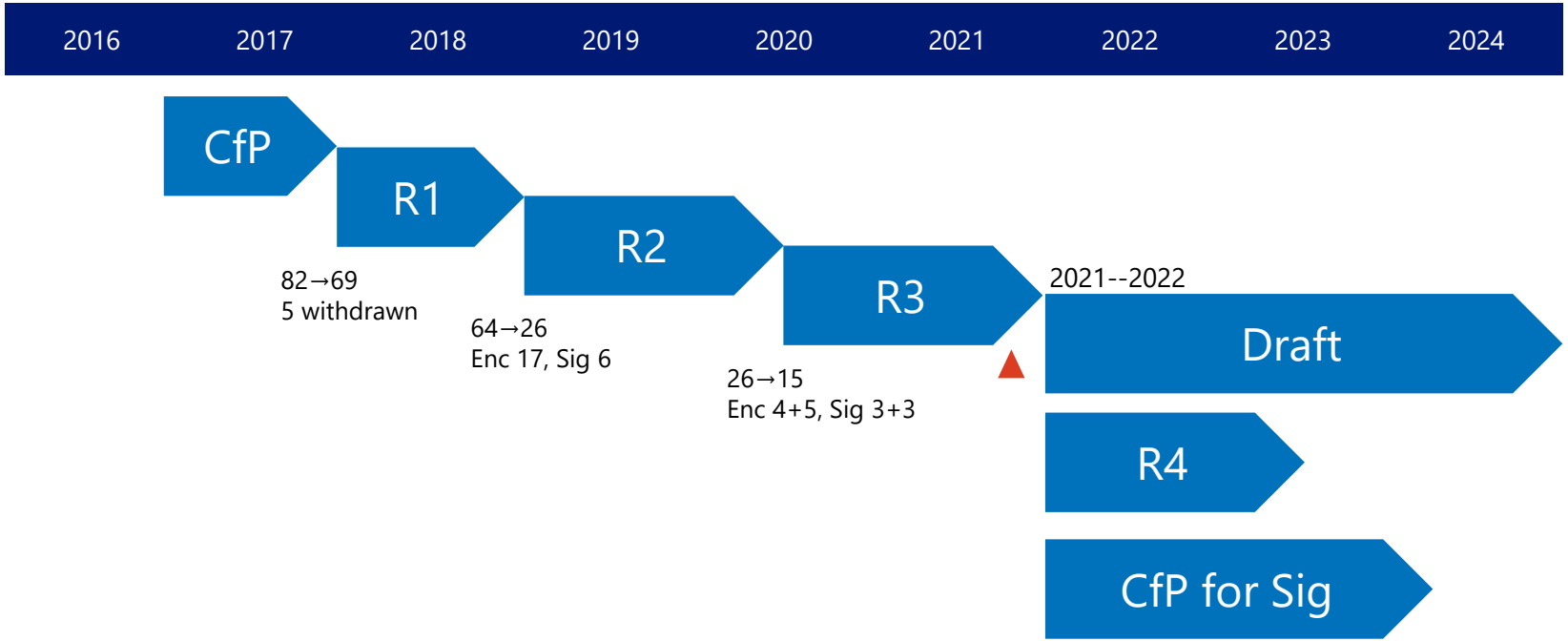
2^{110} T-gates with 2^{34} qubits

1core 3GHz Intel Skylake
Keygen: 2.3 days
Dec: 2.1 hours
Enc: 10.1 minutes

[BFHLV17] Brenstein, Fried, Heninger, Lou, Valenta (NIST PQC Round 1)
[BHLV17] Bernstein, Heninger, Lou, Valenta (PQCRYPTO 2017)

NIST PQC

NIST PQC Standardization Timeline



https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline

NIST PQC Round 1

69 candidates are accepted from 82 submissions

BIG QUAKE, BIKE, CFPKM, Classic McEliece, Compact LWE, CRYSTALS-Dilithium, CRYSTALS-Kyber, DAGS, Ding Key Exchange, DME, DRS, DualModeMS, Edon-K, EMBLEM and R.EMBLEM, Falcon, FrodoKEM, GeMSS, Giophantus, Gravity-SPHINCS, GuessAgain, Gui, Hila5, HiMQ-3, HK17, HQC, KCL, KINDI, LAC, LAKE, LEDAkem, LEDApkc, Lepton, Lima, Lizard, LOCKER, LOTUS, LUOV, McNie, Mersenne-756839, MQDSS, NewHope, NTRUEncrypt, pqNTRUsign, NTRU-HRSS-KEM, NTRU Prime, NTS-KEM, Odd Manhattan, Ouroboros-R, Picnic, Post-Quantum RSA-Encryption, Post-Quantum RSA-Signature, pqsigRM, QC-MDPC KEM, qTESLA, RaCoSS, Rainbow, Ramstake, RankSign, RLCE-KEM, Round2, RQC, RVB, SABER, SIKE, SPHINCS+, SRTPI, Three Bears, Titanium, WalnutDSA

NIST PQC Round 1 (2017-12-21—)

Enc 49:

Code 17: BIG QUAKE, BIKE, Classic McEliece, DAGS, Edon-K, HQC, LAKE, LEDAkem, LEDApkc, Lepton, LOCKER, McNie, NTS-KEM, Ouroboros-R, QC-MDPC KEM, RLCE-KEM, RQC
Lattice 22: Compact LWE, CRYSTALS-Kyber, Ding Key Exchange, EMBLEM and R.EMBLEM, FrodoKEM, Giophantus, Hila5, KCL, KINDI, LAC, Lima, Lizard, LOTUS, NewHope, NTRUEncrypt, NTRU-HRSS-KEM, NTRU Prime, Odd Manhattan, Round2, SABER, Titanium, Three Bears
MQ 3: CFPKM, DME, SRTPI
Isogeny 1: SIKE
Mersenne 2: Mersenne-756839, Ramstake
Others 4: Post-Quantum RSA-Encryption, RVB, GuessAgain, HK17

Sig 22:

Code 3: pqsigRM, RaCoSS, RankSign
Lattice 5: CRYSTALS-Dilithium, DRS, Falcon, pqNTRUsign, qTESLA
MQ 8: DME, DualModeMS, GeMSS, Gui, HiMQ-3, LUOV, MQDSS, Rainbow
Sym/Hash 3: Gravity-SPHINCS, Picnic, SPHINCS+
Others 2: Post-Quantum RSA-Signature, WalnutDSA

NIST PQC Round 2 (2019-01-30—)

Enc 49→17:

Code 17→7: BIG BIKE, Classic McEliece, HQC, LEDAcrypt(=LEDAkem+LEDApkc), NTS-KEM, ROLLO(=LAKE+LOCKER+Ouroboros-R), RQC
Lattice 22→9: CRYSTALS-Kyber, FrodoKEM, LAC, NewHope, NTRU(=NTRUEncrypt+NTRU-HRSS-KEM), NTRU Prime, Round5(=HIla5+Round2), SABER, Three Bears
MQ 3→0:
Isogeny 1→1: SIKE
Mersenne 2→0:
Others 4→0:

Sig 22→9:

Code 3→0:
Lattice 5→3: CRYSTALS-Dilithium, Falcon, qTESLA
MQ 8→4: GeMSS, LUOV, MQDSS, Rainbow
Sym/Hash 3→2: Picnic, SPHINCS+
Others 2→0:

NIST PQC Round 3 (2020-07-22—)

4 Finalists: Enc

Classic McEliece* (code)
CRYSTALS-Kyber (lattice)
NTRU (lattice)
SABER (lattice)

3 Finalists: Sig

CRYSTALS-Dilithium (lattice)
Falcon (lattice)
Rainbow (MQ)

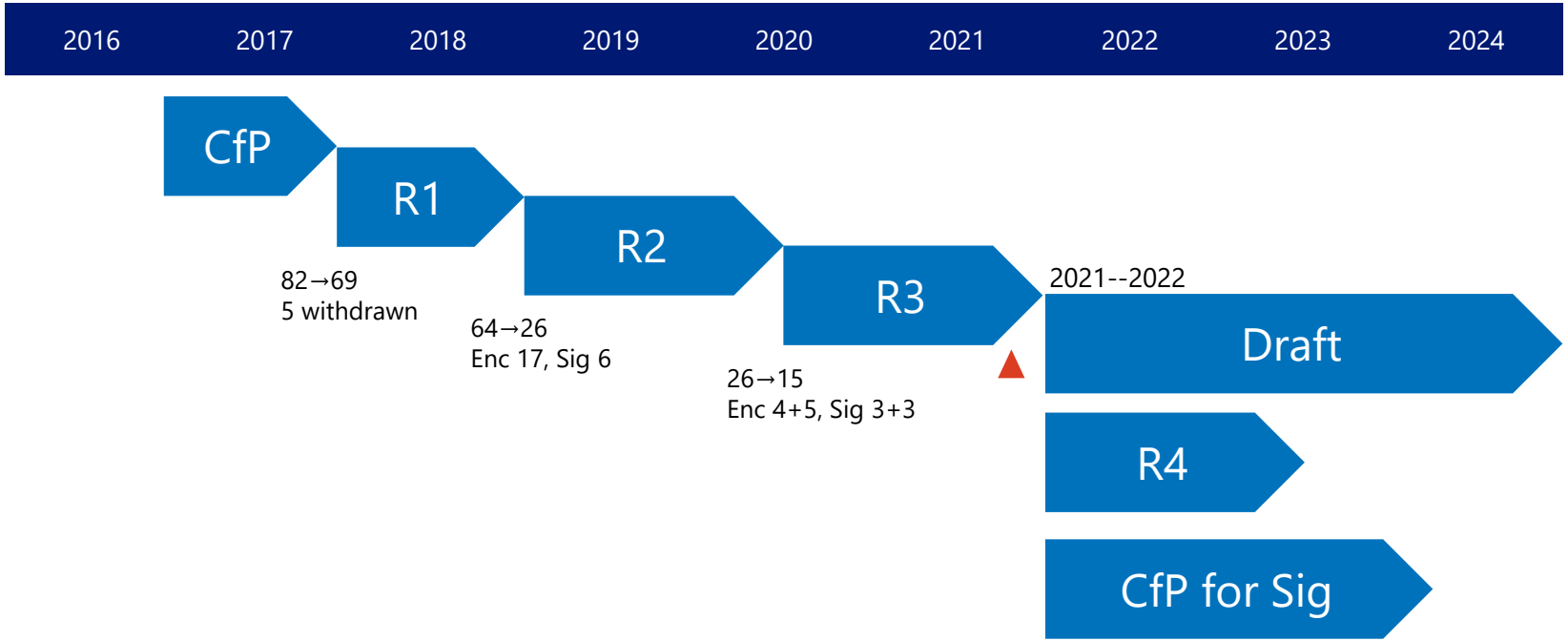
5 Alternates: Enc

BIKE (code)
FrodoKEM (lattice)
HQC (code)
NTRU Prime (lattice)
SIKE (isogeny)

3 Alternates: Sig

GeMSS (MQ)
Picnic (ZK)
SPHICS+ (Hash)

NIST PQC Standardization Timeline

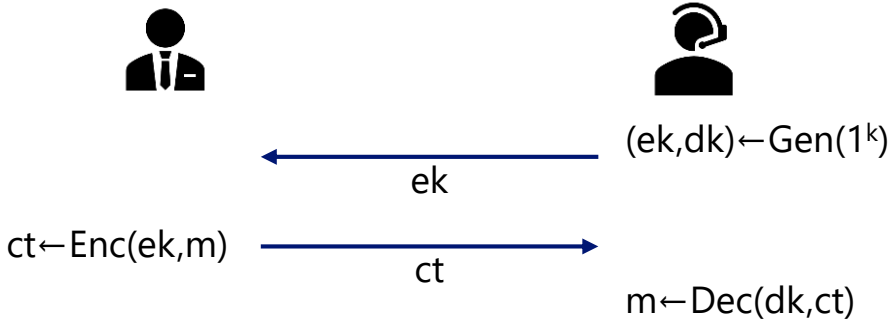


https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline

PKE

PKE – Public-Key Encryption

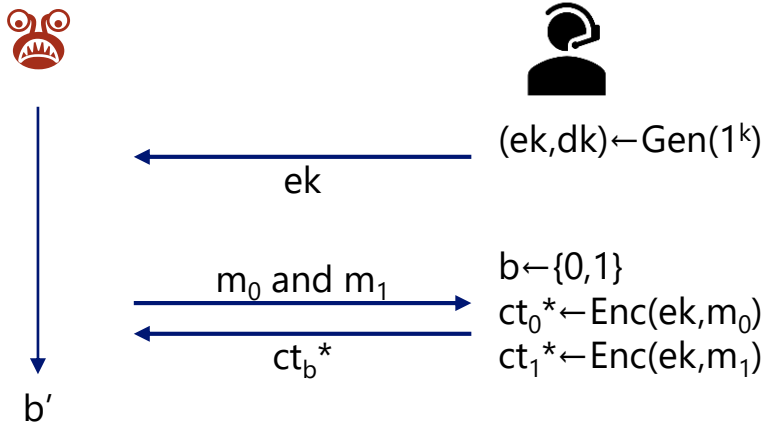
- $\mathrm{Gen}(1^k) \rightarrow (\mathrm{ek}, \mathrm{dk})$
- $\mathrm{Enc}(\mathrm{ek}, \mathrm{m}) \rightarrow \mathrm{ct}$
- $\mathrm{Dec}(\mathrm{dk}, \mathrm{ct}) \rightarrow \mathrm{m}/\perp$



PKE

- $Gen(1^k) \rightarrow (ek,dk)$
- $Enc(ek,m) \rightarrow ct$
- $Dec(dk,ct) \rightarrow m/\perp$

- **IND-CPA:**

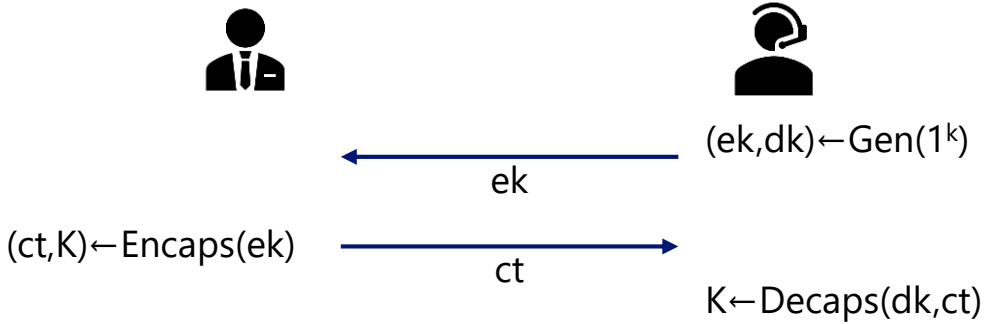


Adv = $|Pr[b=b'] - ½|$
PKE is IND-CPA-secure if Adv is negligible

KEM

KEM – Key Encapsulation Mechanism

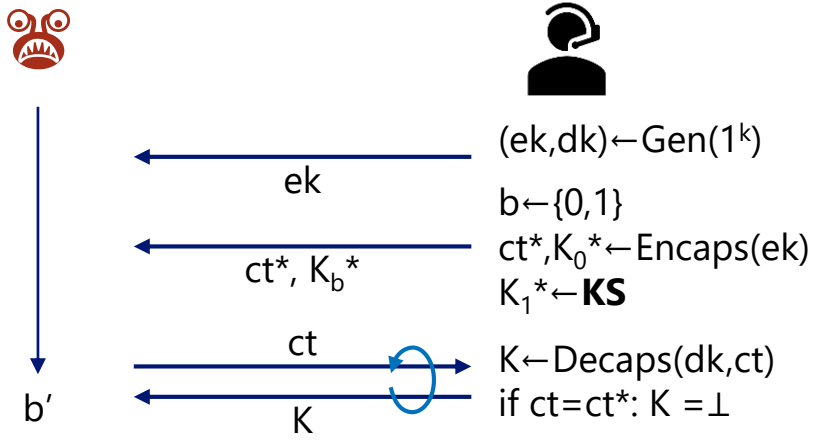
- $\mathrm{Gen}(1^k) \rightarrow (\mathrm{ek}, \mathrm{dk})$
- $\mathrm{Encaps}(\mathrm{ek}) \rightarrow (\mathrm{ct}, K)$
- $\mathrm{Decaps}(\mathrm{dk}, \mathrm{ct}) \rightarrow K/\perp$



KEM

- $Gen(1^k) \rightarrow (ek,dk)$
- $Encaps(ek) \rightarrow (ct,K)$
- $Decaps(dk,ct) \rightarrow K/\perp$

- **IND-CCA:**



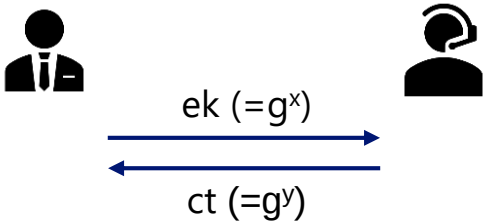
Adv = $|Pr[b=b'] - ½|$

KEM is IND-CCA-secure if Adv is negligible

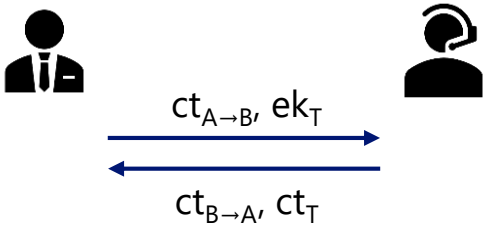
Why IND-CCA?

Key exchange:

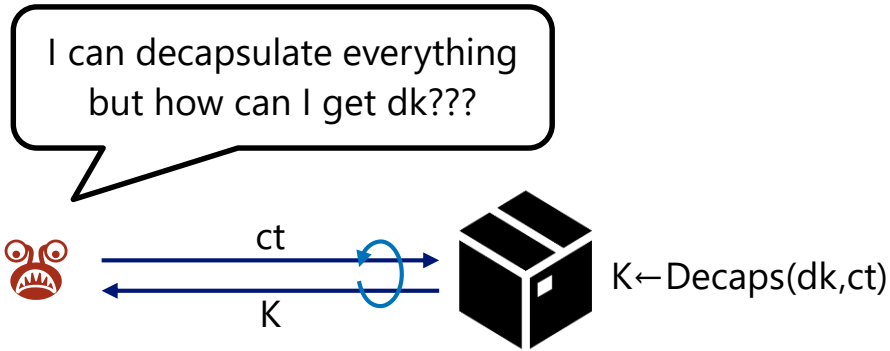
- Safe key reuse



- Authenticated KE



KR-CCA:



How to construct IND-CCA KEM

- **Direct:** hard and inefficient
- **Transformtion:** weak PKE/TDF → IND-CCA KEM
- **T. in StdM**: easy but inefficient (?)
  [...,C:KopWat19,C:KitMatTan19,C:HofKopWat20]
- **T. in ROM**: easy and efficient!
  [CCS:BelRog93,C:FujOka99,JC:FujOka13,...]

(Edit: ROM)

- **StdM:** Hash:$\{0,1\}^* \to \{0,1\}^k$ is given as an algorithm or program
  - 👹 can compute Hash(x) by itself
- **ROM**: Hash:$\{0,1\}^* \to \{0,1\}^k$ is idealized as a random oracle
  - 👹 cannot compute Hash(x)!
  - 👹 asks x to the oracle and obtain Hash(x)
- Note: ∃bad PKE s.t. secure in ROM but insecure in StdM [CGH01]

QROM

Why QROM?

QRO appears in CS theory

QROM for crypt. [AC:BDFLSZ11]

Why QROM?

 can implement hash *by itself*

→  can get $\Sigma_x|x,H(x)\rangle$

→ quantumly-accessible RO

Problems:

1. Seeking the list seems hard!
2. Programming seems hard!

1. see [C:Zhandry19]
2. see e.g. [EPRINT:GHHM20]

Fujisaki-Okamoto Transformation

FO_{KEM}

KEM = FO_{KEM}[PKE,G,H]:

- Encaps(ek;x):
  - ct=Enc(ek,x;G(x))
  - K=H(x)
- Decaps(dk,ct):
  1. x′←Dec(dk,ct)
  2. if Encaps(ek;x′)=ct
  3. then return H(x′)
  4. else return ⊥

Adapted version [C:FujOka99]:

FO_{KEM} is IND-CCA in ROM
if PKE is OW-CPA (and some)

$FO_{KEM}^{!\perp}$ [TCC:HofHovKil17,C:JZCWM18,…]

NTT

KEM = FO_{KEM}[PKE,G,H,H']:

- Encaps(ek;x):
  - ct=Enc(ek,x;G(x))
  - K=H(x)
- Decaps(dk,ct):
  1. x'←Dec(dk,ct)
  2. if Encaps(ek;x')=ct
  3. then return H(x')
  4. else return H'(s,ct) //PRF

New techniques

- [HHK17] Implicit Rejection
- [JZCWM18] New proof tech.
- [SXY18+HKSU20] Using DS
- [JZM19,XY19,LW21] HU, HFO
- [Zhandry19] New QRO sim.

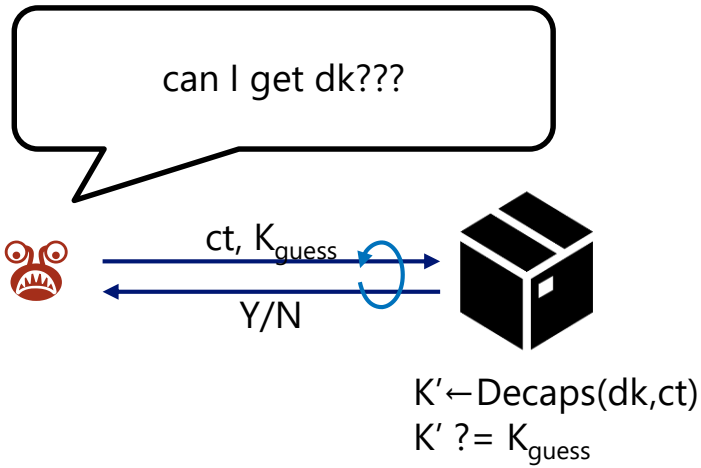
O2Hs

- [AHU19] SC-O2H
- [BHHHP19] DS-O2H
- [KSSSS20] MRM-O2H (Measure&Rewind)

KRA using PCO/KMO/FDO

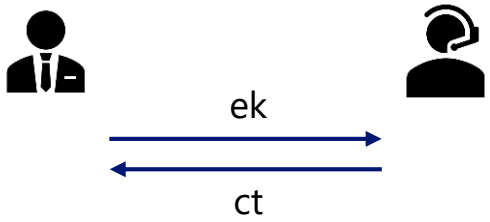
Key-Mismatch Oracle (KMO)

KR-KMA against KEM



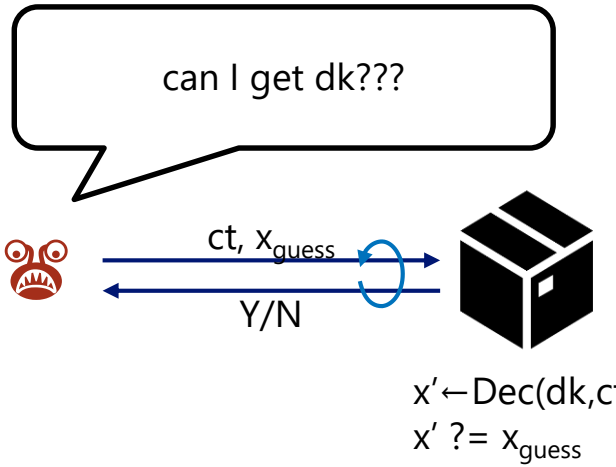
Key-Mismatch Oracle:

Key exchange with fixed secret



Plaintext-Checking Oracle (PCO)

KR-PCA against PKE



Plaintext-Checking Oracle:

KEM w/ FO and KE w/ fixed secret

→ KMO checks $K' = K_{guess}$

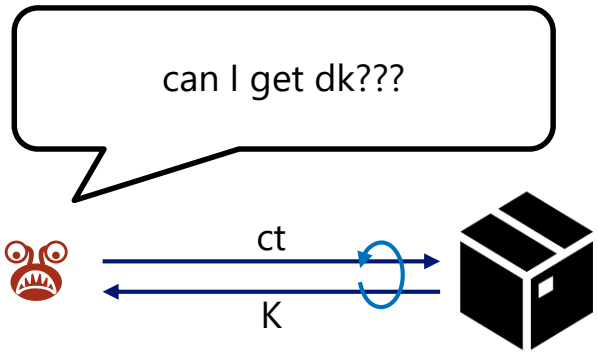
→ We can check $H(x') = H(x_{guess})$

→ We can implement PCO of PKE

IND-PCA [OP01]

Faulty Decapsulation Oracle (FDO)

KR-FDA against PKE/KEM:



x′ ←Dec(dk,ct) ~~K′←Decaps(dk,ct)~~
K = H(x′)

Faulty Decapsulation Oracle:

Decapsulation ignores the validity test

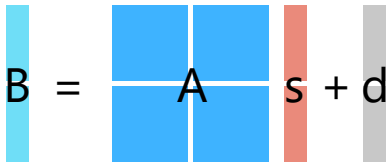
More powerful than PCO

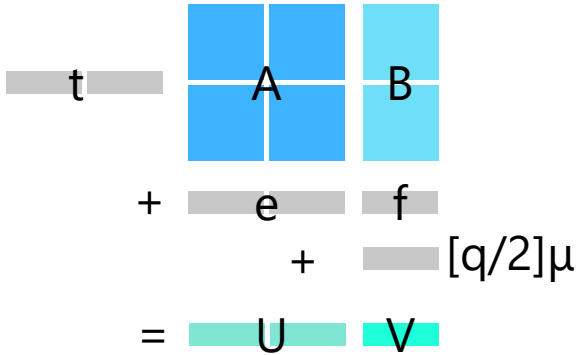
Example: Kyber

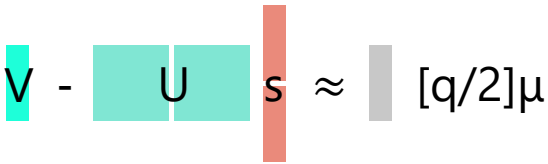
Kyber-512

Kyber-512's PKE:

- $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{256}+1)$
- Ψ_3 = centered bin. dist. over $[-3,+3]^{256}$
- Gen(pp):
  - A←$\mathcal{R}_q^{2\times 2}$, s,d←$\Psi_3^2$, B=As+d
  - ek = (A,B), dk = s
- Enc(ek,μ; t,e,f):
  - U = tA + e, V = tB + f + [q/2]μ
  - ct=(U',V') = (comp(U),comp(V))
- Dec(dk,ct):
  1. (U,V) = (dec(U'), dec(V'))
  2. μ' = near((2/q) (V-Us)) mod 2



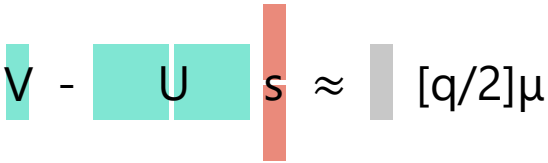




KR-PCA against Kyber512

Kyber-512's PKE:

- $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{256}+1)$
- Ψ_3 = centered bin. dist. over $[-3,+3]^{256}$
- Dec(dk,ct): s←Ψ_3^2
  1. (U,V) = (dec(U'), dec(V'))
  2. μ' = near((2/q) (V-Us)) mod 2



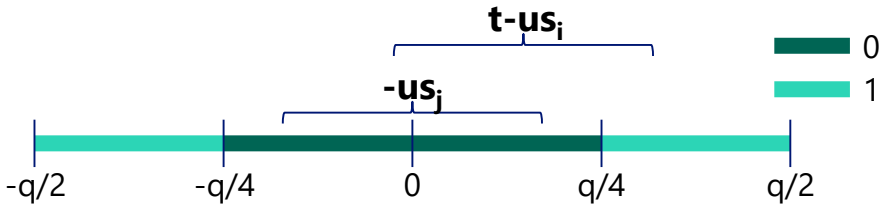
Idea [HV20]:

Consider U=(u,0), V = t x^i

→ μ_i' =near((2/q) (t - u s_i)) mod 2

and μ_j' =near((2/q) (-u s_j)) mod 2

→ Determine s_i by checking μ_i



[HV20] Huguenin-Dumittan and Vaudenay (ACNS2020)

Behaivior of μ_i' w/ u=-276

(a) Kyber512

sk_i \ t	−3	−2	−1	0	1	2	3
−3	1	1	1	0	0	0	0
−2	1	1	0	0	0	0	0
−1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
+1	0	0	0	0	0	0	1
+2	0	0	0	0	0	1	1
+3	0	0	0	0	1	1	1

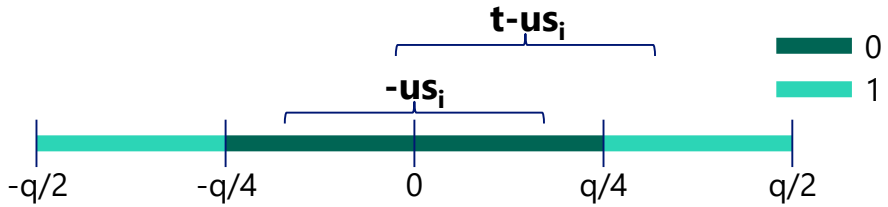
Idea [HV20]**:**

Consider U=(u,0), V = t x^i

→ μ_i' =near((2/q) (t - u s_i)) mod 2

and μ_j' =near(-(2/q) u s_j) mod 2

→ Determine s_i by checking μ_i



[HV20] Huguenin-Dumittan and Vaudenay (ACNS2020)

Survey of KR-PCAs

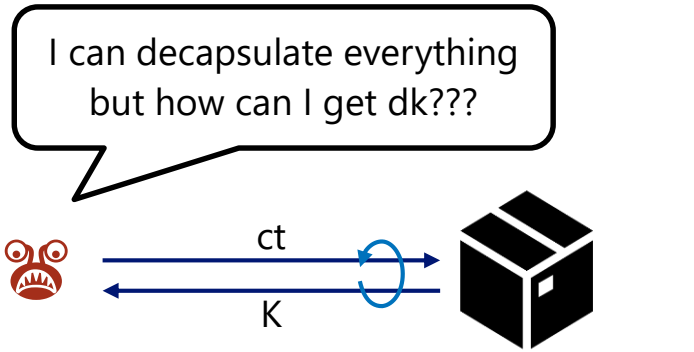
	Ref.
Kyber	[QCD19,RRCB20,HV20,RBRC20,QCZ+21]
Saber	[HV20,OUKT21,NDGJ21,QCZ+21]
FrodoKEM	[BDH+19,RRCB20,VV20,QCZ+21]
NTRU LPRime	[New:XIU+21]
NTRU-HPS	[DDS+19] (There are older ones [JJ00,HS00,…])
NTRU-HRSS	[ZCQD21]
Streamlined NTRU Prime	[REB+21]

See [App.C:XIU+21]

SCA/FIA

Side-Channel/Fault-Injection Analysis

KR-CCA



Side-Channel Analysis

- computation will leak dk
- (power, timing, EM, sound,…)

Fault-Injection Analysis

- make computation faulty
  - › skip instruction
  - › flip memory

Leads to FDO

Survey of KR-PCAs

	Ref.
Kyber	[QCD19,RRCB20,HV20,RBRC20,QCZ+21]
Saber	[HV20,OUKT21,NDGJ21,QCZ+21]
FrodoKEM	[BDH+19,RRCB20,VV20,QCZ+21]
NTRU LPRime	[New:XIU+21]
NTRU-HPS	[DDS+19] (There are older ones [JJ00,HS00,…])
NTRU-HRSS	[ZCQD21]
Streamlined NTRU Prime	[REB+21]

See [App.C:XIU+21]

Survey of KR-SCA/FIA

	Ref.
Kyber	[RRCB20,SKL+20+RR21,XPRO20,BDH+21,PP21]
Saber	[RRCB20,SKL+20+RR21,NDGJ21,NDJ21]
FrodoKEM	[RRCB20,GJN20,SKL+20+RR21]
NTRU LPRime	N/A
NTRU-HPS	[AR21,REB+21] (older ones [SW07,...])
NTRU-HRSS	[AR21,REB+21]
Streamlined NTRU Prime	[HCY19,REB+21]

See [App.C:XIU+21] and [UXT+21]

Fault-Injection Analysis

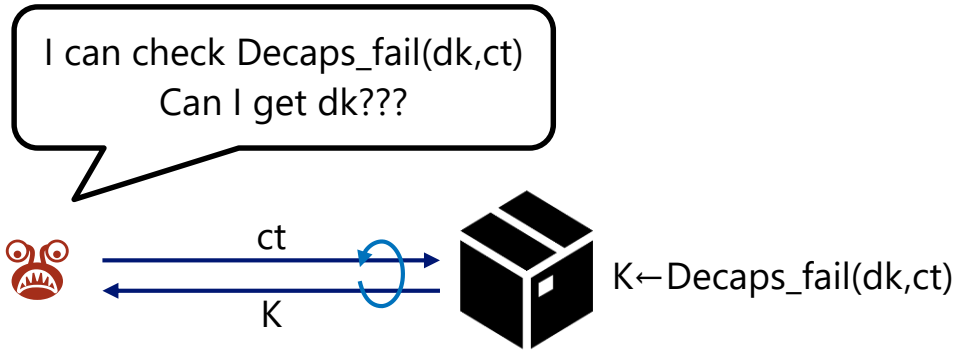
Fault-Injection Analysis

Skip the equality check

- Decaps_fail(dk,ct):
  1. x′←Dec(dk,ct)
  2. ~~if Encaps(pk;x′)=ct~~
  3. then return H(x′)
  4. ~~else return H′(s,ct) //PRF~~

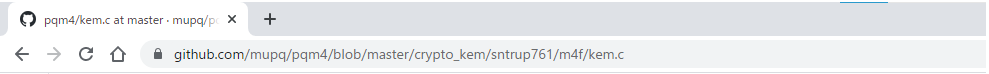
Skip step 2 by injecting power glitch
→We have FDO virtually!

KRA with FDO:



Joint work with Ito, Ueno, Takahashi, and Homma: [EPRINT:XIUTH21] https://ia.cr/2021/840

CCA Bug in NTRU Prime (A.Ito found)



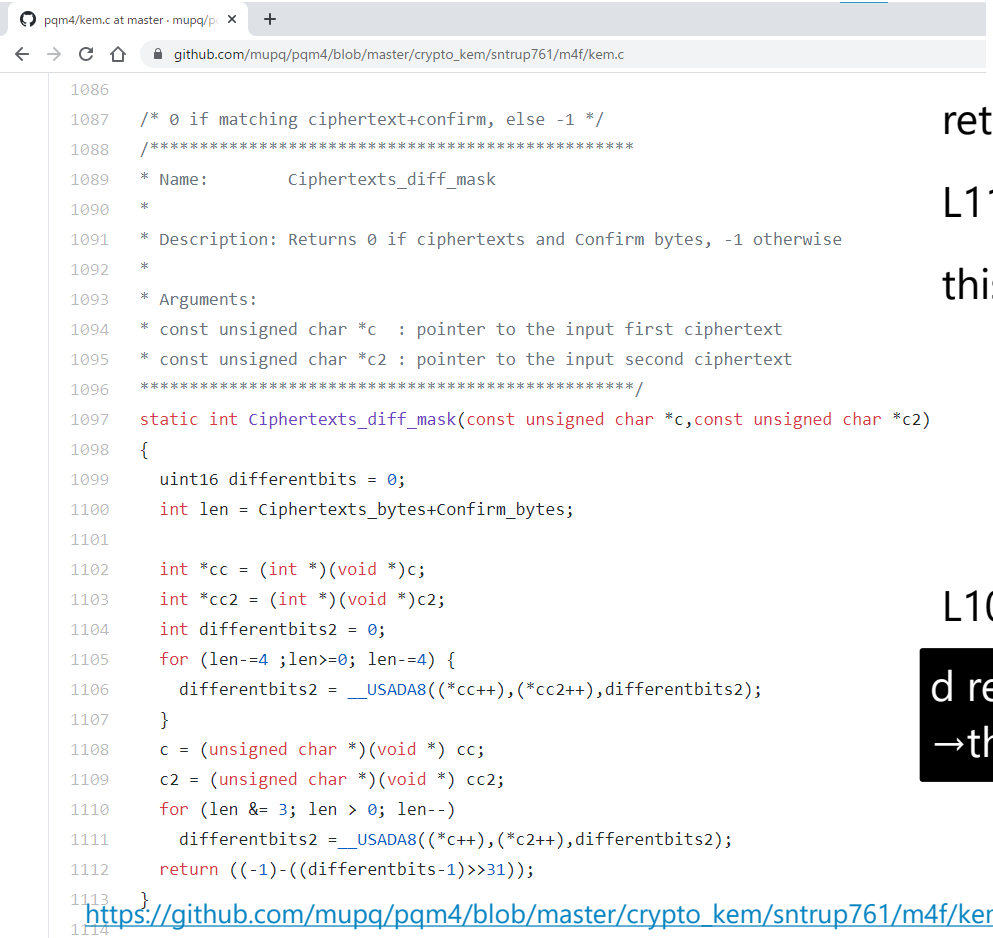

```
1086
1087  /* 0 if matching ciphertext+confirm, else -1 */
1088  /************************************************
1089  * Name:        Ciphertexts_diff_mask
1090  *
1091  * Description: Returns 0 if ciphertexts and Confirm bytes, -1 otherwise
1092  *
1093  * Arguments:
1094  * const unsigned char *c  : pointer to the input first ciphertext
1095  * const unsigned char *c2 : pointer to the input second ciphertext
1096  ************************************************/
1097  static int Ciphertexts_diff_mask(const unsigned char *c,const unsigned char *c2)
1098  {
1099    uint16 differentbits = 0;
1100    int len = Ciphertexts_bytes+Confirm_bytes;
1101
1102    int *cc = (int *)(void *)c;
1103    int *cc2 = (int *)(void *)c2;
1104    int differentbits2 = 0;
1105    for (len-=4 ;len>=0; len-=4) {
1106      differentbits2 = __USADA8((*cc++),(*cc2++),differentbits2);
1107    }
1108    c = (unsigned char *)(void *) cc;
1109    c2 = (unsigned char *)(void *) cc2;
1110    for (len &= 3; len > 0; len--)
1111      differentbits2 =__USADA8((*c++),(*c2++),differentbits2);
1112    return ((-1)-((differentbits-1)>>31));
1113  }
1114
```

Compare c and c2

Return 0 if c = c2, -1 otherwise

https://github.com/mupq/pqm4/blob/master/crypto_kem/sntrup761/m4f/kem.c

CCA Bug in NTRU Prime (A.Ito found)




```
1086
1087  /* 0 if matching ciphertext+confirm, else -1 */
1088  /************************************************
1089  * Name:        Ciphertexts_diff_mask
1090  *
1091  * Description: Returns 0 if ciphertexts and Confirm bytes, -1 otherwise
1092  *
1093  * Arguments:
1094  * const unsigned char *c  : pointer to the input first ciphertext
1095  * const unsigned char *c2 : pointer to the input second ciphertext
1096  ************************************************/
1097  static int Ciphertexts_diff_mask(const unsigned char *c,const unsigned char *c2)
1098  {
1099    uint16 differentbits = 0;
1100    int len = Ciphertexts_bytes+Confirm_bytes;
1101
1102    int *cc = (int *)(void *)c;
1103    int *cc2 = (int *)(void *)c2;
1104    int differentbits2 = 0;
1105    for (len-=4 ;len>=0; len-=4) {
1106      differentbits2 = __USADA8((*cc++),(*cc2++),differentbits2);
1107    }
1108    c = (unsigned char *)(void *) cc;
1109    c2 = (unsigned char *)(void *) cc2;
1110    for (len &= 3; len > 0; len--)
1111      differentbits2 =__USADA8((*c++),(*c2++),differentbits2);
1112    return ((-1)-((differentbits-1)>>31));
1113  }
1114
```

return 0 if c = c2 else -1

L1112: return (-1) – ((d-1)>>31):

this returns 0 if d=0 else -1

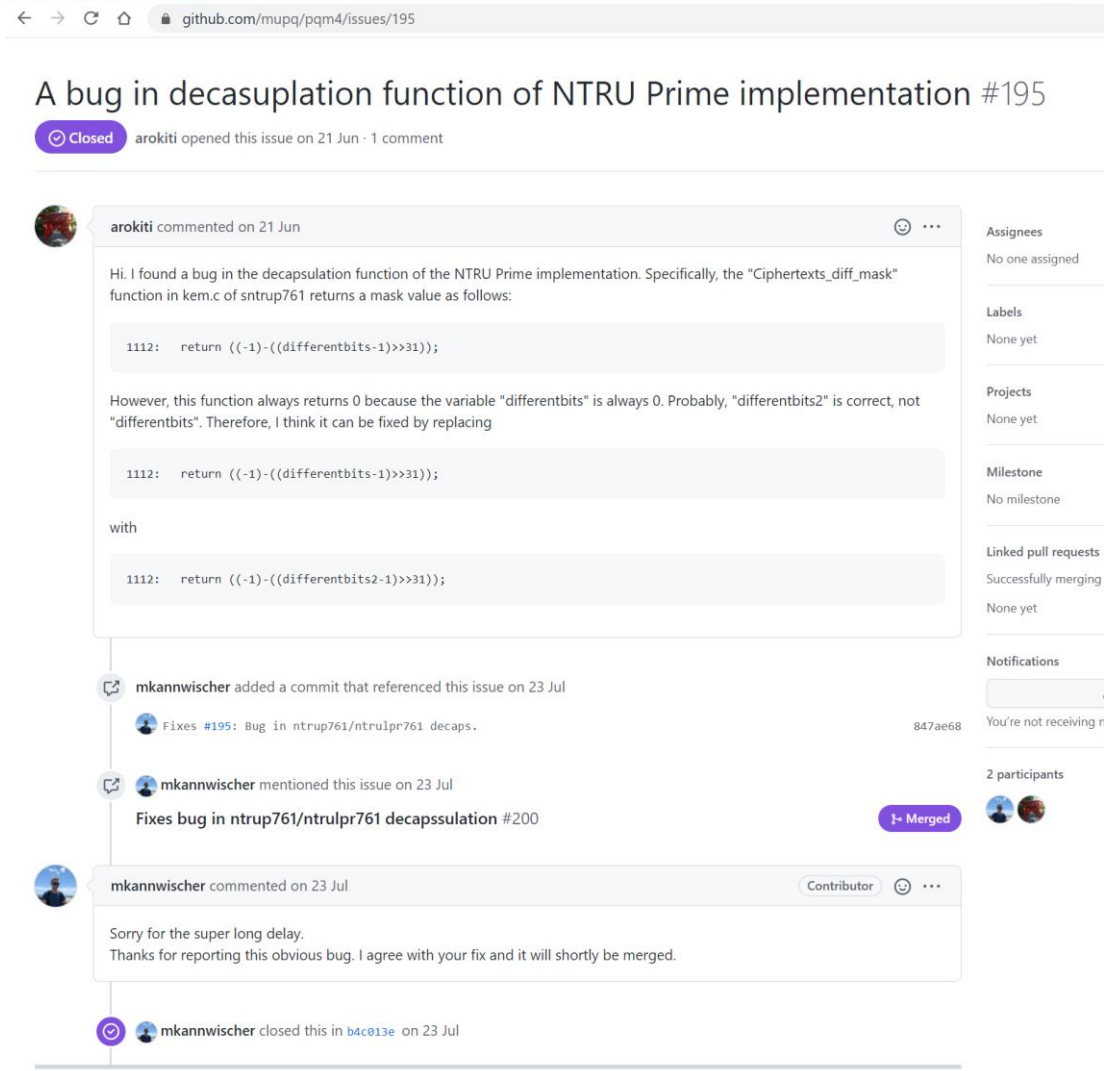
- d-1 = 0xffff_ffff/non-zero
- (d-1)>>31 = -1/0
- (-1) – ((d-1)>>31) = 0/-1

L1099: uint16 d = 0x0000

d remains 0……
→this func. always returns 0!

https://github.com/mupq/pqm4/blob/master/crypto_kem/sntrup761/m4f/kem.c

Fixed!



https://github.com/mupq/pqm4/issues/195

Skip of cmov – Saber, Kyber, NTRU

```
1     int crypto_kem_dec(uint8_t *k, const uint8_t *c, const uint8_t *sk)
2 {
3     uint8_t fail;
4     uint8_t buf[64];
5     uint8_t kr[64]; // Will contain key, coins
6     const uint8_t *pk = sk + SABER_INDCPA_SECRETKEYBYTES;
7     const uint8_t *hpk = sk + SABER_SECRETKEYBYTES - 64;
8                                     // Save hash by storing h(pk) in sk
9
10    indcpa_kem_dec(sk, c, buf);
11    memcpy(buf + 32, hpk, 32);
12    sha3_512(kr, buf, 64);          // kr = G(x',H(pk))
13    fail = indcpa_kem_enc_cmp(buf, kr + 32, pk, c);
14    sha3_256(kr + 32, c, SABER_BYTES_CCA_DEC);    // kr = G1(x',H1(pk)) + H1(ct)
15    cmov(kr, sk + SABER_SECRETKEYBYTES - SABER_KEYBYTES,
16                                                  SABER_KEYBYTES, fail);
17    sha3_256(k, kr, 64);
18    return (0);
19 }
20
```

L15: cmov: kr's half is seed if fail is 0 else $G_1(x',H_1(pk))$
L16: sha3_256: k = $H(G_1(x',H_1(pk)),H_1(ct))$ if fail is 0 else $H(seed,H_1(ct))$

https://github.com/mupq/pqm4/blob/master/crypto_kem/saber/m4fspeed/kem.c

Skip of cmov – Saber, Kyber, NTRU

```
1     int crypto_kem_dec(uint8_t *k, const uint8_t *c, const uint8_t *sk)
2  {
3     uint8_t fail;
4     uint8_t buf[64];
5     uint8_t kr[64]; // Will contain key, coins
6     const uint8_t *pk = sk + SABER_INDCPA_SECRETKEYBYTES;
7     const uint8_t *hpk = sk + SABER_SECRETKEYBYTES - 64;
8                                     // Save hash by storing h(pk) in sk
9
10    indcpa_kem_dec(sk, c, buf);
11    memcpy(buf + 32, hpk, 32);
12    sha3_512(kr, buf, 64);              // kr = G(x',H(pk))
13    fail = indcpa_kem_enc_cmp(buf, kr + 32, pk, c);
14    sha3_256(kr + 32, c, SABER_BYTES_CCA_DEC);   // kr = G1(x',H1(pk)) + H1(ct)
15    cmov(kr, sk + SABER_SECRETKEYBYTES - SABER_KEYBYTES,
16                                                  SABER_KEYBYTES, fail);
17    sha3_256(k, kr, 64);
18    return (0);
19 }
20
```

```
1     bl   sha3_256
2  .LVL26:
3  .loc 1 79 3 is_stmt 1 view .LVU62
4        uxtb    r3, r7
5        add r1, r4, #1536
6        add r0, sp, #64
7        movs    r2, #32
8        bl   cmov
9  .LVL27:
10       .loc 1 82 3 view .LVU63
11       movs    r2, #64
12       mov r0, r6
13       add r1, sp, r2
14       bl   sha3_256
```

L15: cmov: kr's half is seed if fail is 0 else $G_1(x',H_1(pk))$
L16: sha3_256: k = $H(G_1(x',H_1(pk)),H_1(ct))$ if fail is 0 else $H(seed,H_1(ct))$

https://github.com/mupq/pqm4/blob/master/crypto_kem/saber/m4fspeed/kem.c

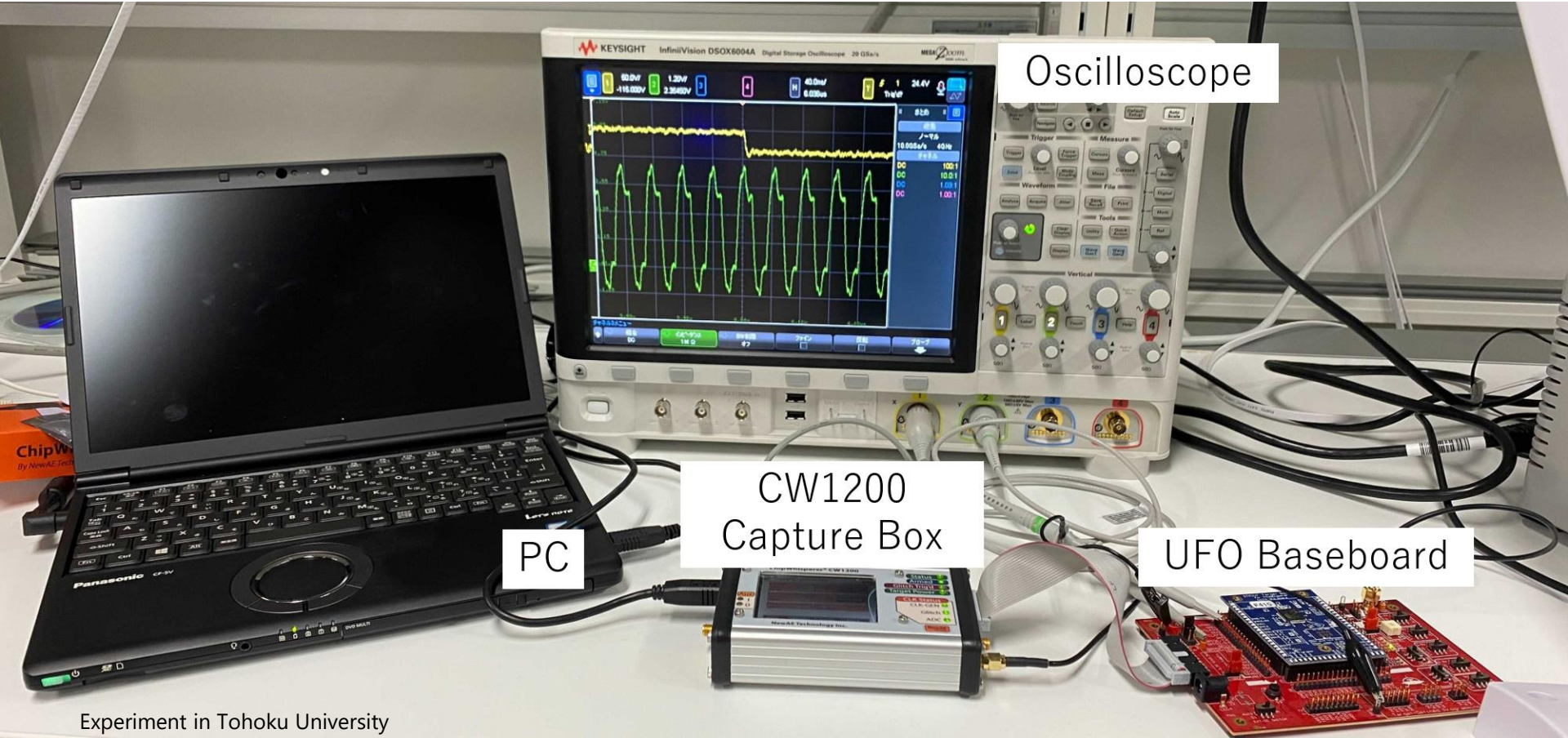
Skip of cmov – Saber, Kyber, NTRU

```
1   int crypto_kem_dec(uint8_t *k, const uint8_t *c, const uint8_t *sk)
2   {
3       uint8_t fail;
4       uint8_t buf[64];
5       uint8_t kr[64]; // Will contain key, coins
6       const uint8_t *pk = sk + SABER_INDCPA_SECRETKEYBYTES;
7       const uint8_t *hpk = sk + SABER_SECRETKEYBYTES - 64;
8                                   // Save hash by storing h(pk) in sk
9
10      indcpa_kem_dec(sk, c, buf);
11      memcpy(buf + 32, hpk, 32);
12      sha3_512(kr, buf, 64);              // kr = G(x',H(pk))
13      fail = indcpa_kem_enc_cmp(buf, kr + 32, pk, c);
14      sha3_256(kr + 32, c, SABER_BYTES_CCA_DEC);   // kr = G1(x',H1(pk)) + H1(ct)
15      cmov(kr, sk + SABER_SECRETKEYBYTES - SABER_KEYBYTES,
16                                                   SABER_KEYBYTES, fail);
17      sha3_256(k, kr, 64);
18      return (0);
19  }
20
```

```
1       bl   sha3_256
2   .LVL26:
3   .loc 1 79 3 is_stmt 1 view .LVU62
4       uxtb    r3, r7
5       add r1, r4, #1536
6       add r0, sp, #64
7       movs    r2, #32
8       bl   cmov
9   .LVL27:
10      .loc 1 82 3 view .LVU63
11      movs    r2, #64
12      mov r0, r6
13      add r1, sp, r2
14      bl   sha3_256
```

L15: cmov: kr's half is seed if fail is 0 else $G_1(x',H_1(pk))$
L16: sha3_256: k = $H(G_1(x',H_1(pk)),H_1(ct))$ if fail is 0 else $H(seed,H_1(ct))$
By skipping L8 in RHS, k = $H(G_1(x',H_1(pk)),H_1(ct))$ *always*

Experiment



Experiment in Tohoku University

Experiment results

100 skip trials / each trial takes 0.1s

	# failures	# successes	# exp. queries
Kyber512	60	52	5908
LightSaber	74	46	15515
ntruhps2048509	33	33	2235
ntrulpr653	100	100	1306

SCA

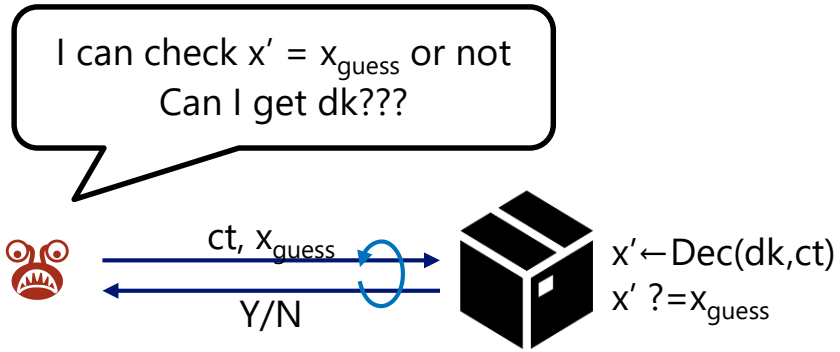
Side-Channel Analysis

Get info. From Re-Enc

- Decaps_leak (dk,ct):
  1. x′←Dec(dk,ct)
  2. if Encaps(pk;x′)=ct
  3. then return H(x′)
  4. else return H′(s,ct) //PRF

Encaps(pk;x′) evaluates G(x′), which leaks x′
→We have PCO/KMO virtually!

KRA with PCO:



Joint work with Ito, Ueno, Takahashi, and Homma: [EPRINT:XIUTH21] https://ia.cr/2021/840

SCA (setting)

Table 5: Experimental implementations and numbers of used traces

	Non-protected AES/SHAKE software	Non-protected AES hardware	Masked bit-sliced AES software	Masked AES hardware based on TI
Reference	`pqm4` [KRSS19, pqm21]	SASEBO IP [Toh]	Schwabe and Stoffelen [SS16, git21]	Ueno *et al.* [UHA17]
Device	STM32F415RGT6	Xilinx Kintex-7	STM32F407VGT6U	Xilinx Kintex-7
Board	NewAE Technology STM32F	SAKURA-X	STM32F407G-DISC1	SAKURA-X
Side-channel trace	Supply voltage current	Supply voltage current	EM radiation	Supply voltage current
Measurement interface	NewAE technology chip-whisperer CW308	On-board coaxial connector	Langer EMV-Technik RF-U T-2 probe	On-board coaxial connector
Oscilloscope	Keysight Technologies MSOX6004A			
# Training traces	30,000	30,000	900,000	980,000
# Validation traces	10,000			
# Test traces	10,000			

SCA based on NN

Distinguish fixed vs. random

	Accuracy	# of traces for $1-10^{-6}$ Majority	# of traces for $1-10^{-6}$ Likelihood
AES SW and SHAKE SW	0.998	5	2
AES HW	0.999	5	2
Masked AES SW	0.960	11	5
Masked AES HW (TI)	0.515	> 5000	> 5000

Theoretical estimation

	# exp. queries non-masked imple.	# exp. Queries masked SW
Kyber512	3072	7680
LightSaber	6144	15360
ntruhps2048509	2036	5090
ntrulpr653	2612	6530

Agenda

- Why PQC?
- NIST PQC
- PKE and KEM
- Key-recovery attack using oracles
- Fault-injection analysis and Side-channel analysis

Joint work with 田中裕太郎・伊東燦・上野嶺・本間尚文（東北大） 高橋順子（NTT社会情報研究所）
[XIUTH21] Xagawa, Ito, Ueno, Takahashi, Homma (ASIACRYPT 2021) https://eprint.iacr.org/2021/840
[UXTITH21] Ueno, Xagawa, Tanaka, Ito, Takahashi, Homma (TCEHS Vol. 2022, Issue 1) https://eprint.iacr.org/2021/849