(Most) Post-Quantum Bugs are just Plain Old Bugs

Markku-Juhani O. Saarinen
<mjos@mjos.fi>

Lab 1339 / teserakt.io

P.O. Box 1339, CB1 0BZ, Cambridge, UK

CTCrypt 2018 Rump Session

Suzdal, Russia – May 28, 2018

NIST Post-Quantum Cryptography Project

An attempt to identify new **Public Key Encryption** and **Digital Signature** algorithms that are resistant to attacks with quantum computers of the future.

- **20.12.2016**: Call for proposals released.
- **30.11.2017**: Candidate submission deadline.
- **21.12.2017**: ~~82~~ *69* submissions accepted.
- **11-13.04.2018**: Standardization conference.

  – – *We are here. Candidates are out.* – –
- **2018 / 2019**: Round 2 begins.
- **2020 / 2021**: Round 3 begins.
- **2022 / 2024**: Draft standards available.

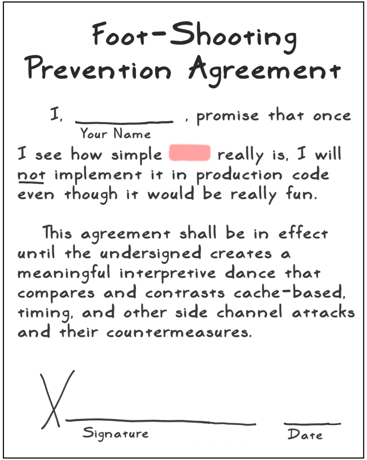
First Round PQC Candidates: Analysis is ongoing..

BIG QUAKE, *BIKE*, **CFPKM**, *Classic McEliece*, **Compact LWE**, *CRYSTALS-DILITHIUM*, *CRYSTALS-KYBER*, **DAGS**, *Ding KEX*, **DME**, *DRS*, *DualModeMS*, **Edon-K**, *EMBLEM*, *FALCON*, *FrodoKEM*, *GeMSS*, **Giophantus**, *Gravity-SPHINCS*, **Guess Again**, *Gui*, **HILA5**, *HiMQ-3*, **HK17**, *HQC*, *KINDI*, *LAC*, *LAKE*, **LEDAkem**, **LEDApkc**, **Lepton**, *LIMA*, *Lizard*, *LOCKER*, *LOTUS*, *LUOV*, **McNie**, *Mersenne-756839*, *MQDSS*, *NewHope*, *NTRUEncrypt*, *NTRU-HRSS-KEMf*, *NTRU Prime*, *NTS-KEM*, *Odd Manhattan*, *OKCN/AKCN/CNKE*, *Ouroboros-R*, *Picnic*, *pqNTRUSign*, *pqRSA encryption*, *pqRSA signature*, **pqsigRM**, *QC-MDPC KEM*, *qTESLA*, *RaCoSS*, *Rainbow*, *Ramstake*, **RankSign**, **RLCE-KEM**, *Round2*, *RQC*, **RVB**, *SABER*, *SIKE*, *SPHINCS+*, **SRTPI**, *Three Bears*, *Titanium*, **WalnutDSA**

14 broken or withdrawn. 7 amended with tweaks.

Ugh, that's 3165 pages of specs.

Also, 100+ C implementations!



Useful checks #1: If I decrypt that, do I get the same message back ?

NTRU KEM 1024, a rather prominent candidate, **can't decrypt what it encrypts**.

`NTRUEncrypt/Reference_Implementation/ntru-kem-1024/NTRUEncrypt.c`:

```
    /* extract the last bit of rh */
    for (i=0;i<LENGTH_OF_HASH*2;i++)
    {
        seed[i] = (rh[i*8] & 1);
        for (j=1;j<8;j++);
        {
            seed[i] <<= 1;
            seed[i] += (rh[i*8+j] & 1);
        }
    }
```

Can you spot the bug ? This one created a compiler warning. There probably others, as we still don't have a patch for this. The KAT test vectors are of course useless too.

Useful checks #2: How many bits does my secret key actually have ?

If you claim "*n*-bit security", then your secret key should have *at least* ***n*** bits, right ?

.. so I implemented a simple **bit-bias entropy tester** for KEM shared secrets ..

.. and some candidates **failed it**. For example AKCN-MLWE has only 248 bits of classical security because ..

```
$ grep "ss = " OKCN_AKCN_CNKE/KAT/kem/AKCN-MLWE/PQCkemKAT_288.rsp
ss = 6FD6A1CB8BB6C649ED78B252158C08E9FBB26BE98866B59C18A6746DF3C85700
ss = 2E7C87B16678DA8E9218D17EF717D8ABC5271F610B63A3A34A3F50A814646300
ss = 996C9EF62A0D10C288364B649A2725D5EA752AAA8EB1A2E60FB5AC06B6BFAB00
                                   .. the last byte is always zero ^^
```

OKCN_AKCN_CNKE/Reference_Implementation/kem/AKCN-MLWE/ref/parameter.h:

```
#define Z_SEED_BYTES         32
#define MATRIX_SEED_BYTES    32
#define NOISE_SEED_BYTES     31  // Why ?! (Works if I set fhis to 32)
```

Useful checks #3: Try Flipping a Random Bit!

Alice		Bob
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}()$	$\xrightarrow{\mathsf{PK}}$	
	$\xleftarrow{\mathsf{CT}}$	$(\mathsf{CT}, \mathsf{K}) \leftarrow \mathsf{Encaps}(\mathsf{PK})$
$\mathsf{K} \leftarrow \mathsf{Decaps}(\mathsf{SK}, \mathsf{CT})$		

We tested **flipping a single bit** in PK or CT and observing the difference in K.

Alice

K_A

$\xrightarrow{\mathsf{PK}}$

$\xleftarrow{\mathsf{CT} \oplus 2^j}$

Mallory

$\approx$

$\xrightarrow{\mathsf{PK} \oplus 2^i}$

$\xleftarrow{\mathsf{CT}}$

Bob

K_B

In **AKCN** and **OKCN** candidates the Hamming distance (K_A, K_B) was often **one bit**..

Useful checks #4: Where does my algorithm spend all of its time ?

This happened to me. In NIST benchmarking, **HILA5** spends **90 %** of its time in the *only part of the code that is not mine*, the **stupid random number generator**.

`FinalAPIdocs09252017/rng.c` from NIST:

```
if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors(); // allocate memory

if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_ecb(), NULL, key, NULL))
    handleErrors();            // (full key schedule - 95% of time here)

if(1 != EVP_EncryptUpdate(ctx, buffer, &len, ctr, 16))
    handleErrors();            // (single block operation, 16 bytes)

ciphertext_len = len;          // (both are unused, but hey I'm NIST)

EVP_CIPHER_CTX_free(ctx);      // (destroy expanded key for no reason)
```

HILA5 was **7.0**× faster after I replaced this with a version that does **NOT** re-do full key schedule for each output block. Significant speedups also for other candidates.

So how about those timing attacks and other side-channel features ?

Only a minority of designers have considered side-channel attacks at all.

Implementation attacks will remain the easiest way to break PQC algorithms too.