

The case for Fujisaka-Okamoto decapsulation key checks

Daniel Shiu

The Fujisaka-Okamoto transform

The CCA security of many modern cryptographic primitives is dependent on the Fujisaka-Okamoto (FO) transform.

F-O recreates key generation to ensure that keys are not corrupted nor faultily/maliciously generated.

In ML-KEM F-O checks encapsulator keys, but not decapsulator keys.

Public key poisoning

This leaves ML-KEM vulnerable to malformed decapsulator keys. We focus on the issue of public key poisoning.

In many public key schemes, public keys are polymorphic in that different public keys can be substituted while preserving correctness.
e.g textbook RSA $N \equiv 120N$, EC El Gamal $(G, A) \equiv (rG, rA)$.

Depending on sanitisation checks, this can lead to vulnerabilities such as Coppersmith, key misbinding, small subgroup attacks, Curveball etc.

LWE is polymorphic

Public $A, \mathbf{t}$, private $\mathbf{s}$

$$\mathbf{t} = A \circ \mathbf{s} + \mathbf{e}$$

Let $\mathbf{t}' = \mathbf{t} + \mathbf{e}'$ then

$$\mathbf{t}' = A \circ \mathbf{s} + \underbrace{(\mathbf{e} + \mathbf{e}')}_{\mathbf{e}''}$$

same $\mathbf{s}$ but $\mathbf{e}'' \gtrapprox \mathbf{e}$.

More noise; more failures

The Ding key exchange $\mathbf{t}_A \circ \mathbf{s}_B \approx \mathbf{s}_A \circ \mathbf{t}_B$ succeeds if

$$-\frac{q}{4} < \mathbf{e}_A \circ \mathbf{s}_B - \mathbf{s}_A \circ \mathbf{e}_B < \frac{q}{4}$$

“Poisoning” $\mathbf{e}_A$ makes failures more common;

failures leak information about $\mathbf{s}_A$ and $\mathbf{e}_A$ to adversaries who know $\mathbf{e}_B$ and $\mathbf{s}_B$.

ML-KEM768 as intended

Success if all 768 coefficients satisfy

$$-833 < \underbrace{\mathbf{e}_A \circ \mathbf{s}_B}_{\approx \mathcal{N}(0,768)} - \underbrace{\mathbf{s}_A \circ (\mathbf{e}_B + \mathbf{r}_B)}_{\approx \mathcal{N}(0,1478)} + \underbrace{r}_{\approx U(-104,104)} < 833$$

Failure is a $\approx 15\sigma$ event $\approx 7.4 \times 10^{-51}$.

Just add poison...

Replace $\mathbf{t}_A$ with $\mathbf{t}_A + (100, 100, 100, 0, 0, 0, 0, \dots)$

...equivalent to replacing $\mathbf{e}_A$ with $\mathbf{e}_A + (100, 100, 100, 0, 0, 0, 0, \dots)$

$$\begin{aligned}
 -833 &< \underbrace{\mathbf{e}_A \circ \mathbf{s}_B}_{\approx \mathcal{N}(0,768)} - \underbrace{\mathbf{s}_A \circ (\mathbf{e}_B + \mathbf{r}_B)}_{\approx \mathcal{N}(0,1478)} + \\
 &+ \underbrace{r}_{\approx U(-104,104)} + (100, 100, 100, 0, 0, 0, 0, \dots) \circ \mathbf{s}_B < 833
 \end{aligned}$$

Almost never?

Given coefficients where $\mathbf{s}_B = (2, 2, 2, *, *, *, *, \dots)^T$ and $r \approx 100$, failure becomes a $\approx 3\sigma$ event.

Similarly for underflow

Conditions on $\mathbf{s}_B$ and r roughly 2^{-15} event (though an attacker can prefilter).

Extracting information

Given a failure and knowledge of s_B and r , the overflowing coefficient can be identified. A contribution of around 133 is still needed from the other terms:

$$e_{A,j} \circ s_{B,j}$$

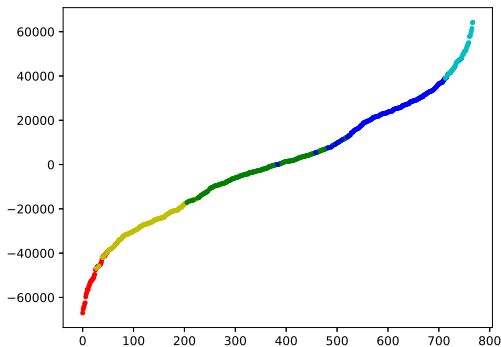
and

$$s_{A,j} \circ (e_{B,j} + r_{B,j})$$

We know the blue values and uses them to score the unknown red values.

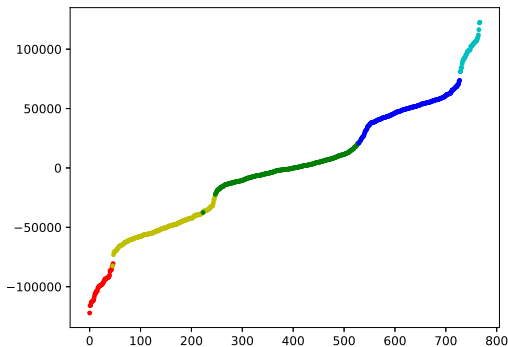
How good is the score?

Scoring e_A from around 2500 failures; coefficients sorted by score; colour denotes actual coefficient



How good is the score? II

Scoring s_A from around 2500 failures; coefficients sorted by score; colour denotes actual coefficient



Recap

1. Legitimate encapsulator correctly generates s_A and t_A , but does not protect the integrity of the public value t_A
2. Malicious user swaps t_A for “poisoned” t'_A ; system still works almost always
3. Malicious user submits encapsulated values likely to trigger decapsulation failures due to poison.
4. Use the failures to score and recover s_A using a few thousand failures.

How to design against this

The F-O can be used to defend against malformed public values, but in ML-KEM only encapsulators' keys are checked using F-O.

Regenerating decapsulator key material from its seed prevents this. ML-KEM *allows* but does not *require* this. In particular it is not possible if the seed is not retained.

It is proposed that HQC private keys are stored as seeds, enforcing regeneration of public values and preventing poisoning.