Fault-enabled chosen-ciphertext attacks on Kyber

Julius Hermelink[1,2] Peter Pessl[2] Thomas Pöppelmann[2]

[1]Universität der Bundeswehr München

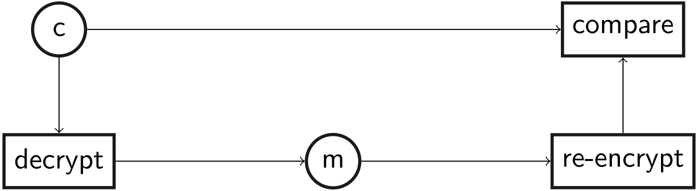
[2]Infineon Technologies AG

Preliminaries - Kyber

In this work, we present an attack on Kyber using the combination of a chosen-ciphertext attack and a single-bit fault attack.

- Kyber is a CCA2-secure post-quantum KEM.
- Finalist in the NIST standardization process.
- Relies on the hardness of the MLWE problem (lattice-based).
- Three parameter sets: Kyber512, Kyber768, Kyber1024.
- Built from an underlying PKE using a variant of the FO-transform.
- Works in $R = \mathbb{F}_q[x]/(x^n - 1)$ and R^k.

Kyber - Decapsulation



During decryption, the message is recovered from the polynomial[1][2]

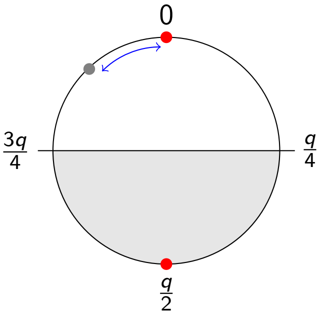
$$\begin{aligned} rec &= m + \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e_1} + e_2 \\ &= m + noise. \end{aligned}$$

where **e**, **s** are secret, other terms are known to the attacker, and the noise is small.

[1] Ignoring compression errors
[2] Vectors of polynomials in bold

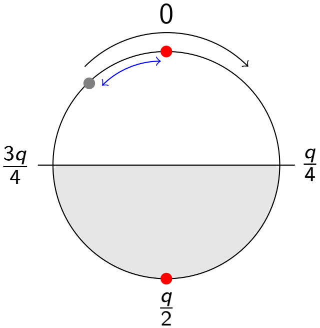
Kyber - Message recovery

- Encryption: 0-bits mapped to 0, 1-bits mapped to $\frac{q}{2}$ (one bit to one coefficient).
- Decryption: Recover from $rec = m + noise$ by mapping to 0 if closer to 0 than to $\frac{q}{2}$, otherwise to 1.
- Upper half of the circle mapped to 0, lower half to 1.



Decryption errors

- Message is recovered from $rec = m + noise$.
- Adding $\frac{q}{4}$ to a coefficient of rec: Corresponding message bit might change depending on which side the noisy message is on.
- Decryption error happens if $noise$ coefficient is positive.

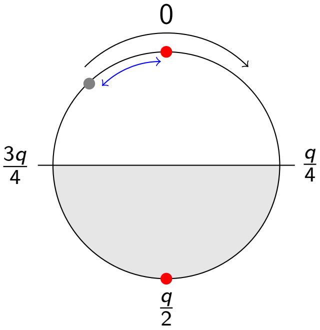


Decryption errors

- Adding $\frac{q}{4}$ and observing decryption errors tells us if a coefficient of

$$noise = \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e_1} + e_2.$$

  is positive or negative[3].
- Gives inequalities involving secrets $\mathbf{e}, \mathbf{s}$.



[3] Ignoring compression errors

Pessl and Prokop's attack

A recent fault attack by Pessl and Prokop takes advantage of decryption errors.

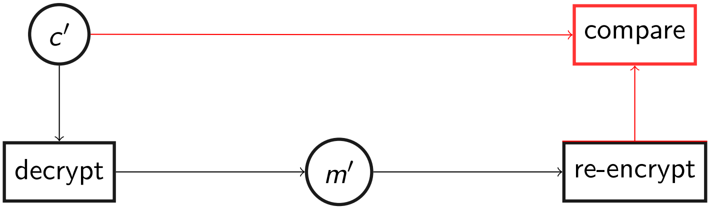
- Pessl and Prokop fault the decoder to cause the addition of $\frac{q}{4}$.
- From each fault/decapsulation: Recover one inequality.
- Solve inequalities by updating distributions of coefficients using obtained inequalities.

Several limitations:

- Prevented by shuffling.
- Very specific fault model.
- Depends on the implementation.

Our attack

- Send manipulated ciphertext c' with $\frac{q}{4}$ added to one coefficient of a valid ciphertext c.
- Device under attack obtains c'' from re-encryption.
- After decryption, fault one bit of stored ciphertext c' to match c.
- Thereby, the FO-transform effectivly compares c against c''.
- Observe decryption errors and obtain inequalities.



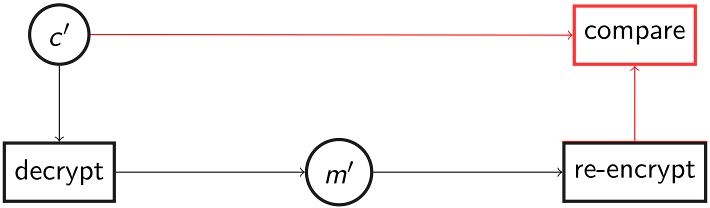
Our attack

By introducing the fault

- The device decrypts c', which is c with a $\frac{q}{4}$-error added in one coefficient.
- Result of re-encryption c'' is compared against c (as c' was corrected).

Two cases:

1. c' causes decryption error $\Rightarrow$ decrypt returns $m' \neq m \Rightarrow c'' \neq c$.
2. c' causes no decryption error $\Rightarrow$ decrypt returns $m' = m \Rightarrow c'' = c$.



Fault model

Fault location in time:

- After decrypt was called,
- and before the re-encryption comparison.

Value to be faulted:

- Either the stored ciphertext,
- or the ciphertext obtained from re-encryption.

Fault model: Set, reset, or flip a single bit.

Why one-bit faults

But why are one-bit faults sufficient?

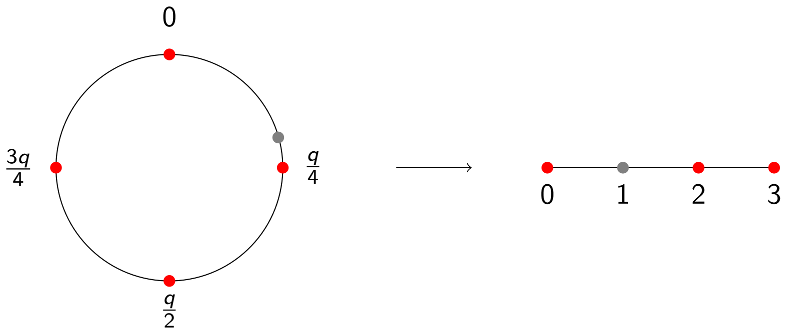
- Ciphertexts are compressed before being outputed.
- Compression is lossy and changing one bit corresponds to adding a multiple of $\frac{q}{2^d}$ (rounded up or down).
- Recovering messages from noisy versions is compression with $d = 1$.

$$\text{compress}(x, d) = \left\lceil (2^d/q) \cdot x \right\rfloor$$
$$\text{decompress}(x, d) = \left\lceil (q/2^d) \cdot x \right\rfloor$$

Kyber compression

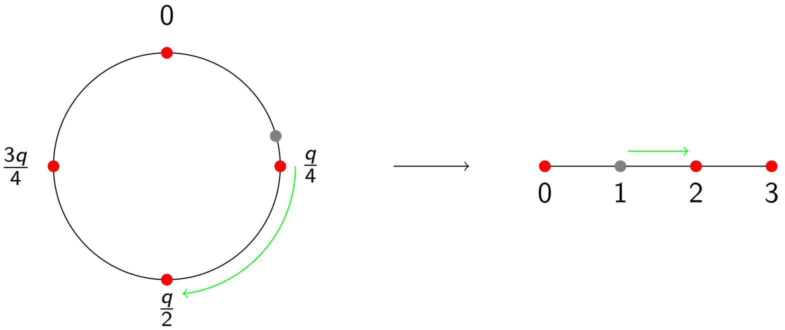
Compression of a coefficient x can be thought of as

- having 2^d points on a circle,
- choosing the closest one to x,
- returning the index of that point.



Kyber compression

Decompression maps a value $i \in \{0 \ldots 2^d - 1\}$ to the i-th point on the circle. Changing one bit on the right (compressed), corresponds to an addition (or substraction) of a multiple of $\frac{q}{2^d}$ (rounded up or down) on the left (uncompressed).



Solving inequalities

The first steps of our attack are to create, send, and then correct chosen ciphertexts.

- We then obtain a number of inequalities.
- Those inequalities give information about the private key, but how to solve them?

We tried

- using integer linear programming to solve for the private key,
- using lattice reduction with the framework provided by Dachman-Soled et al [4].

Our attempts seemed to be computationally very expensive and were not successful.

[4] https://eprint.iacr.org/2020/292

Solving inequalities

Simply solving a system of inequalities ignores additional information we have.

- The unknown variables are the coefficients of the secrets **e** and **s**.
- Those were sampled from a binomial distribution.

Pessl and Prokop developed a different approach:

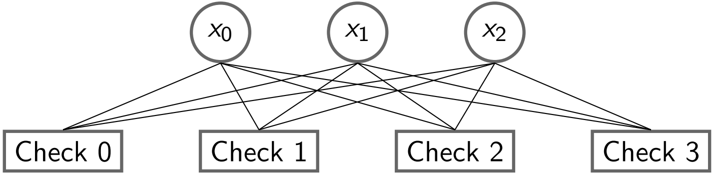
- Initialize a vector of proabilitiy distributions representing each unknown coefficient.
- In each step update using the inequality matrix.

This solves the system of inequalities with a reasonable number of inequalities and time. Unfortunately, the memory consumption is rather high.

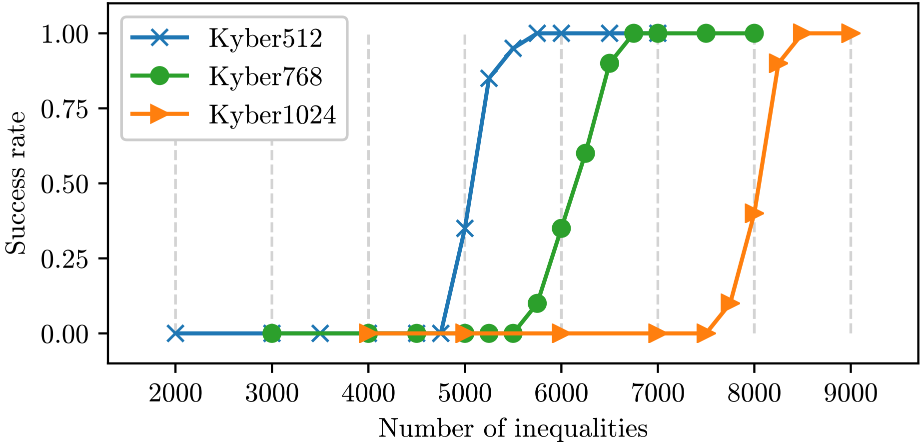
Solving inequalities using belief propagation

Using belief propagation needs 20-30% less faults and requires significantly less RAM.

- For each unknown coefficient of x (given by $\mathbf{e}$ and $\mathbf{s}$), we initialise a *variable node* with a prior given by the binomial distribition they were sampled from.
- For each inequality, we initialise a *check node*.



Performance - Success rate



Perfomance - Runtime

Runtimes in minutes on an Intel(R) Xeon(R) Gold 6242 with 32 and 8 threads.

Parameter set	Iterations	32 threads	8 threads
Kyber512 (6000 inequalities)	6.8	3.25	9.3
Kyber768 (7000 inequalities)	6.75	6.7	18.6
Kyber1024 (9000 inequalities)	9	16.9	39.25

Conclusion

Our approach: Instead of sending a valid ciphertext and then applying a fault, send manipulated ciphertext and use a fault to correct.

Several advantages:

- Manipulation is performed offline, therefore observing successful decapsulation means that the fault worked (even with unreliable faults).
- Not prevented by shuffling the decoder and several other countermeasures.
- Fault may be introduced at several places in time/memory over a very long time-span.
- Less implementation specific.

We also present a more efficient recovery method using belief propagation.

Thank you for listening!