Secret in OnePiece: Single-Bit Fault Attack on Kyber

Jian Wang[1,2], Weiqiong Cao[1,3], Hua Chen[1], Haoyuan Li[3]

[1] Trusted Computing and Information Assurance Laboratory,
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3] Zhongguancun Laboratory, Beijing, China

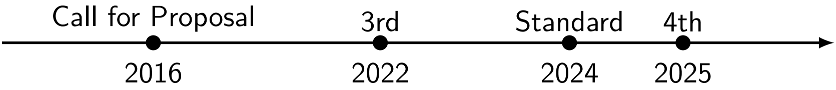
August 13, 2025

Outline

NIST PQC Standardization

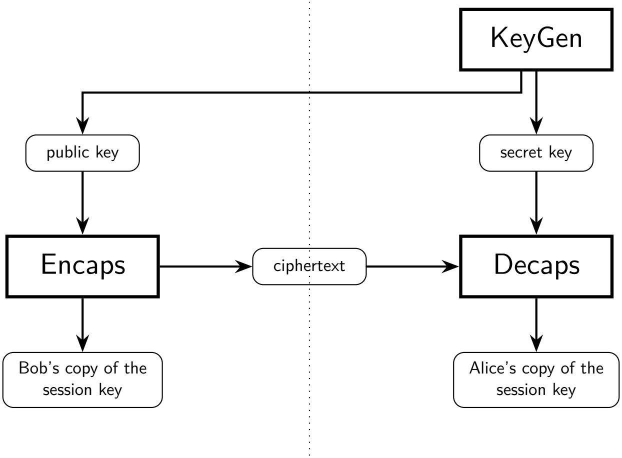
1. Selected algorithms
   - CRYSTALS-Kyber (FIPS-203, ML-KEM)
   - CRYSTALS-Dilithium (FIPS-204, ML-DSA)
   - FALCON
   - SPHINSC+ (FIPS-205, SLH-DSA)
   - HQC (Round4)

2. History



Kyber KEM

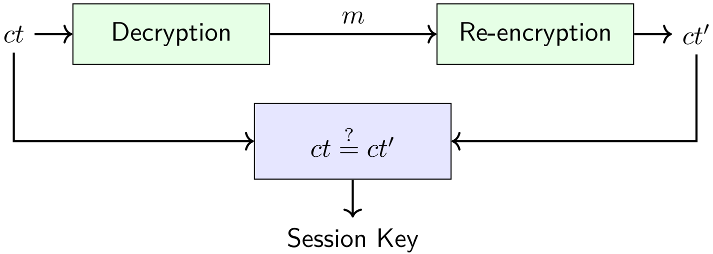
- Key-Encapsulation Mechanism



Related works I

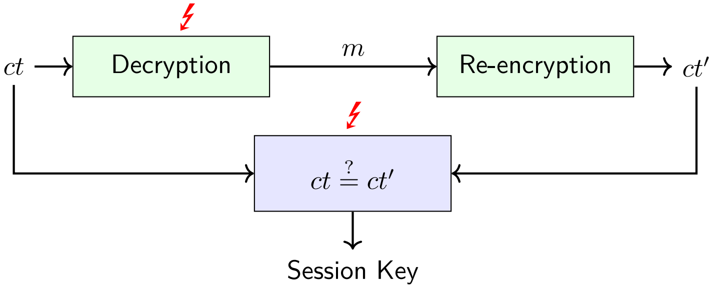
Fujisaki-Okamoto transform

- FO transform → CCA Security
- The CCA-secure decapsulation consists of a decryption, a re-encryption and a ciphertext equality checking.
- FO transform can be regarded as a **redundancy** countermeasure, making traditional fault attacks nearly infeasible.



Related works II

- Fault Attacks on Kyber
  - Injecting faults to disrupt equality checks, enabling chosen-ciphertext attacks [XIU+21].
  - Injecting faults and observing decapsulation success or failure to infer secret-key information [PP21, Del22].



Motivation

- Masked Kyber
  - Conversion between arithmetic and Boolean masking greatly **complicates** the implementation.
  - **Randomness** introduced by masking may aid fault attacks, [Del22] was the first to explore this, proposing an attack on linear operations.
  - The added complexity may enlarge the **attack surface**.
  - [BGR+21] proposed an **arbitrary-order** masked Kyber and a new message decoder.
  - This work builds on [BGR+21] to investigate fault-attack **risks** from masked **nonlinear components**.

Message Decoding

- Decryption
  - **Arithmetic**: compute $mp = v_l - \mathbf{s} \circ \mathbf{u}_l$;
  - **Decoding**: map the noisy polynomial mp to the message m.

Algorithm KyberKEM.Decaps

Require: ciphertext c
Require: private key $sk = (\mathbf{s}, pk, h, z)$
Ensure: session key K

$m \leftarrow$ KyberPKE.Dec$(\mathbf{s}, c)$
$(K, r) \leftarrow G(m, h)$
$\bar{K} \leftarrow J(z||c)$
$c' \leftarrow$ KyberPKE.Enc(pk, m, r)
if $c' \neq c$ **then**
  $K \leftarrow \bar{K}$
end if
return K

Algorithm KyberPKE.Dec

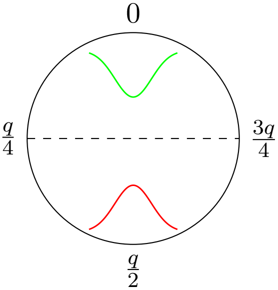
Require: private key $\mathbf{s}$
Require: ciphertext $c = \{c_1, c_2\}$
Ensure: message m

$\mathbf{u}_l \leftarrow \text{Decompress}_{d_u}(\text{Unpack}(c_1))$
$v_l \leftarrow \text{Decompress}_{d_v}(\text{Unpack}(c_2))$
$mp \leftarrow v_l - \mathbf{s} \circ \mathbf{u}_l$
$m \leftarrow$ Decode(mp)
return m

- Message encoding/decoding

$$\text{Enocde}(m) = \begin{cases} \lceil \frac{q}{2} \rfloor, & \text{if } m = 1 \\ 0, & \text{otherwise} \end{cases}$$

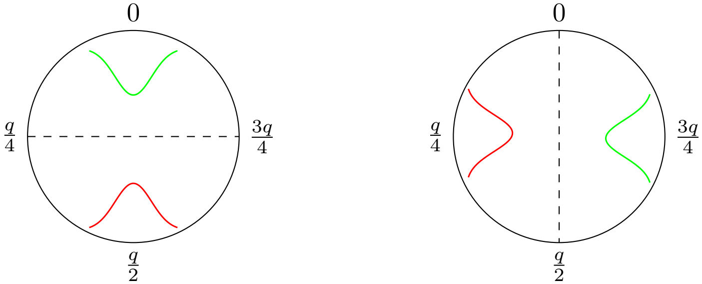
$$\color{red}{\text{Deocde}}(z) = \begin{cases} 1, & \text{if } z \in [\frac{q}{4}, \frac{3q}{4}] \\ 0, & \text{otherwise} \end{cases}$$



In Kyber, $q = 3329$, $\lceil \frac{q}{2} \rfloor = 1665$.

Masking Message Decoder I

- Basic workflow
  1. **Add offset**: increase z by $\frac{3q}{4}$;



  2. **Decode**: check if $z \geq 1665 \; (\lceil q/2 \rfloor)$.

$$\text{Decode}^{s}(z) = \neg z_{11} \oplus (\neg z_{11} \cdot z_{10} \cdot z_{9} \cdot (z_{8} \oplus (\neg z_{8} \cdot z_{7})))$$

Masking Message Decoder II

- Detailed implementation
  - A2B, SecAND, SecXOR, SecREF, Bitslice

Algorithm Masked Decoder

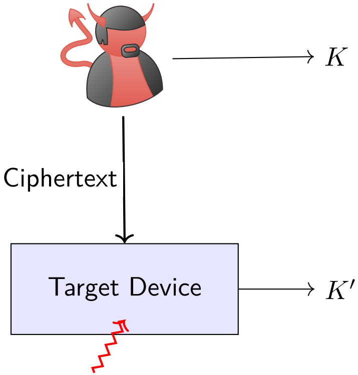
Require: $a^{(\cdot)A}$, $a \in \mathbb{Z}_q[X]$.
Ensure: $m'^{(\cdot)B}$, $m' = \text{Decode}(a) \in \mathbb{Z}_{2^{256}}$.

for $i \leftarrow 0$ to $n-1$ **do**
  $a_i^{(0)A} = a_i^{(0)A} + \left\lfloor \frac{3q}{4} \right\rfloor \mod q$
  $a_i^{(\cdot)B} = \text{A2B}(a_i^{(\cdot)A})$
end for
$z^{(\cdot)B} = \text{Bitslice}(a^{(\cdot)B})$
$m'^{(\cdot)B} = \text{SecAND}(\text{SecREF}(\neg z_8^{(\cdot)B}), z_7^{(\cdot)B})$
$m'^{(\cdot)B} = \text{SecREF}(\text{SecXOR}(m'^{(\cdot)B}, z_8^{(\cdot)B}))$
$m'^{(\cdot)B} = \text{SecAND}(m'^{(\cdot)B}, z_9^{(\cdot)B})$
$m'^{(\cdot)B} = \text{SecAND}(m'^{(\cdot)B}, z_{10}^{(\cdot)B})$
$m'^{(\cdot)B} = \text{SecAND}(m'^{(\cdot)B}, \neg z_{11}^{(\cdot)B})$
$m'^{(\cdot)B} = \text{SecXOR}(m'^{(\cdot)B}, \neg z_{11}^{(\cdot)B})$
return $m'^{(\cdot)B}$

Outline

Attacker model

- What can an attacker do?
  1. Perform encapsulation or trigger decapsulation as needed.
  2. Inject faults during decapsulation.
  3. Observe the session key to detect decapsulation failures.



- Observation on masked decoding
  - Only $z_7 \ldots z_{11}$ are involved.

$$\text{Decode}^s(z) = \neg z_{11} \oplus \big(\neg z_{11} \cdot \neg(z_{10} \cdot z_9 \cdot (z_8 \oplus (\neg z_8 \cdot z_7)))\big)$$

  - Analyze the result of bit flipping, using z_{10} as an example.
    1. If $z_{11} = 1$, the decoding result is fixed at 0, flipping z_{10} will not impact the decoding result.
    2. If $z_9 = 0$ or $(z_8 \oplus (\neg z_8 \cdot z_7)) = 0$, the decoding result is fixed at 0.
    3. Recursive analysis yields the following cases:

z_{10}	z_9	z_8	z_7	Interval of z	d
1	1	0	1	[1664, 2048)	$0 \to 1$
1	1	1	0	[1792, 2048)	
0	1	0	1	[640, 1024)	$1 \to 0$
0	1	1	0	[768, 1024)	

Fault Analysis II

Interval Indication from Fault-Injected Decapsulation

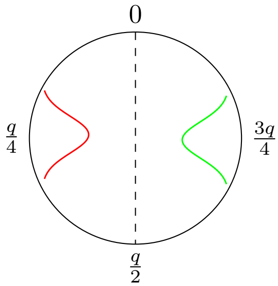
1. Analysis of all 5 bits:

	Decapsulation Failure	Decapsulation Success
z_{11}	$[0, 1792) \cup [2048, 3329)$	$[1792, 2048)$
z_{10}	$[640, 1024) \cup [1664, 2048)$	$[0, 640) \cup [1024, 1664)$
z_9	$[1152, 2048)$	$[0, 1152)$
z_8	$[1664, 1792)$	$[0, 1664) \cup [1792, 2048)$
z_7	$[1792, 1920)$	$[0, 1792) \cup [1920, 2048)$

2. Can we make use of all this information?

Fault Analysis III

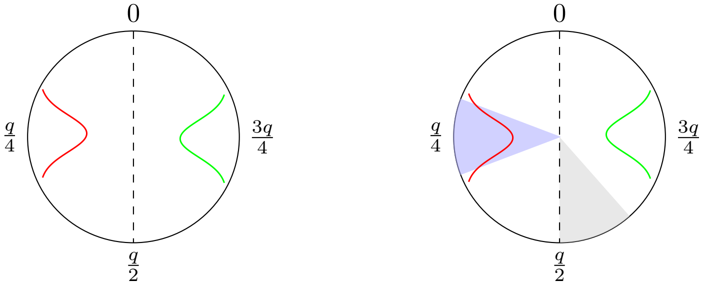
- Usability of interval information
  - A decoded coefficient can be expressed as $m * \lceil \frac{q}{2} \rfloor + \delta$.



  - The probability of the coefficient falling within a certain range can be estimated from the distribution of noise.
  - A decapsulation failure occurs after flipping z_8, then $z \in [1664, 1792)$.
  - However, this event has a very low probability of $2^{-103.9}$.

Fault Analysis IV

- Only z_{10} is suitable as a target.
  - If flipping z_{10} causes decapsulation failure:
    1. $z \in [640, 1024)$, with probability of $1 - 2^{-6.8}$ ($\approx 99.1\%$)
    2. $z \in [1664, 2048)$, with probability $2^{-39.8}$
  - Set the target bit to 1 to ensure the decoded coefficient lies in $[640, 1024)$ when failure occurs, implying $\delta \in [-192, 192)$.



- The system of inequalities

1. The decoded noisy polynomial

$$
\begin{aligned}
mp &= v + \Delta v - (\mathbf{u} + \Delta \mathbf{u}) \circ \mathbf{s} \\
&= \mathbf{t} \circ \mathbf{r} + e_2 + \Delta v - (\mathbf{A} \circ \mathbf{r} + \mathbf{e_1} + \Delta \mathbf{u}) \circ \mathbf{s} + m * \lceil q/2 \rfloor \\
&= \mathbf{r} \circ \mathbf{e} - (\mathbf{e}_1 + \Delta \mathbf{u}) \circ \mathbf{s} + e_2 + \Delta v + m * \lceil q/2 \rfloor. \\
&= \delta + m * \lceil q/2 \rfloor
\end{aligned}
$$

When $\delta \in [-192, 192)$, a decapsulation failure is observed, resulting in a **positive inequality**; otherwise, a **negative inequality**.

2. Repeat ω times to obtain a system of inequalities:

$$
\mathbf{Mx}+\mathbf{b} = \begin{pmatrix} (\mathbf{r})_{(0)}, -(\mathbf{e}_1 + \Delta \mathbf{u})_{(0)} \\ \cdots \\ (\mathbf{r})_{(\omega-1)}, -(\mathbf{e}_1 + \Delta \mathbf{u})_{(\omega-1)} \end{pmatrix} \begin{pmatrix} \mathbf{e} \\ \mathbf{s} \end{pmatrix} + e_2 + \Delta v \underset{\notin}{\in} [-192, 192)
$$

Attack Description II

Solving systems of inequalities

1. Initialize the distribution of secret coefficients:

$$\text{Example:}\{-2 : \frac{1}{16}, -1 : \frac{4}{16}, 0 : \frac{6}{16}, 1 : \frac{4}{16}, 2 : \frac{1}{16}\}$$

2. Update the distribution using inequalities.
   The update rule for the k-th candidate of the j-th coefficient with the i-th inequality is:

$$\mathsf{P}[i,j,k] =$$

$$\Pr\left(-192 \leq \mathbf{M}[i,j](k-\eta_1) + \left(\sum_{j' \in [0,\psi-1] \setminus \{j\}} \mathbf{M}[i,j'] \circ \mathbf{x}[j']\right) + \mathbf{b}[i] < 192\right)$$

3. After all iterations, select candidates with the highest probabilities as predictions.

Attack Description III

- Quick solver
  - Performance bottleneck: convolution operations

$$\mathbf{M}[i, j'] \circ \mathbf{x}[j']$$

  - Approximate $\mathbf{Mx} + \mathbf{b}$ as a normal distribution X via the Central Limit Theorem, with mean μ and standard deviation σ.
  - Convert X to standard normal distribution Z.

$$\mathsf{P}[i, j, k] \approx \Pr\left(\frac{-192 - \mu}{\sigma} \leq Z < \frac{192 - \mu}{\sigma}\right)$$

  - Compute probabilities efficiently using the standard normal cumulative distribution function:

$$\mathsf{P}[i, j, k] \approx F_{norm}(\frac{192 - \mu}{\sigma}) - F_{norm}(\frac{-192 - \mu}{\sigma})$$

Attack Description IV

Challenges in the Solving Process

1. Since δ centers around zero, most candidate values cause decryption failure, making many inequalities weak in narrowing down the possibilities.
2. The collected inequalities are highly imbalanced (e.g., $99 : 1$), which reduces the effectiveness of the solver.

Enhancing Attack Effectiveness via Inequality Filtering

1. Filter 1: Discard low-contribution inequalities offline by selecting ciphertext elements $(\Delta v + e_2)[i]$ near the boundary ± 192.
2. Filter 2: Improve inequality balance by rejection sampling, discarding a proportion α of positive inequalities.

Outline

Fault Injection on Masked Implementation I

❶ Bit flipping via bit setting

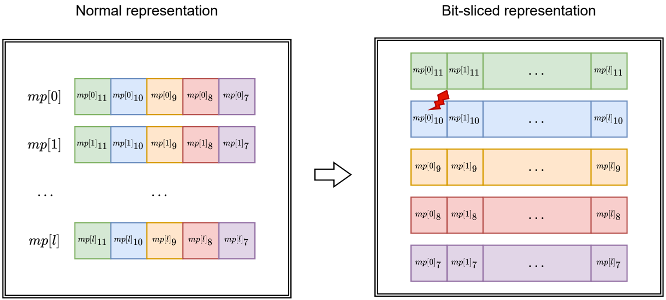
- In Boolean masking, **fixing** $z_{10}^{(i)}$ **to 0 or 1** can induce a bit flip in z_{10} with some probability.
- **Repeat this process** β **times**. If no failure occurs, then with probability $1 - 2^{-\beta}$, $z \notin [640, 1024)$; otherwise, $z \in [640, 1024)$. Only negative inequalities may incur errors under this strategy.

❷ Feasible fault injection

Fault model	Injection Target
Bit-Flip	A2B Bitslice SecAND Load/Store
Stuck-at 0/1	
Instruction Skip	Bitslice

Fault Injection on Masked Implementation II

- Bit flipping via instruction skipping
  - In bit-sliced implementations, skipping an assignment instruction can effectively induce the desired fault:
    1. Instruction skipping → Bit setting
    2. Bit setting → Bit flipping



Outline

Simulation Experiments

Assessment of key recovery and error tolerance

1. Recovering the secret key requires about 30, 000, 540, 000 and 240, 000 inequalities for Kyber512, Kyber768 and Kyber1024, respectively.
2. Error rates up to 30% are tolerable, causing only a moderate increase in required inequalities.

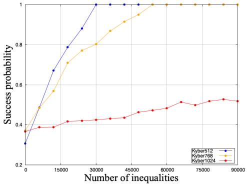


Figure: Solving filtered inequalities for all three security levels.

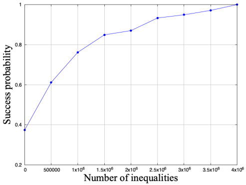


Figure: Solving filtered inequalities for Kyber1024 with $\alpha = 0.94$.

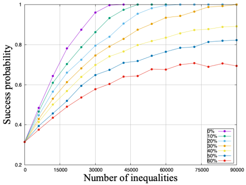
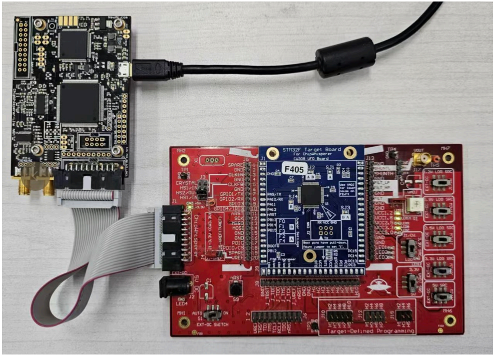


Figure: Solving corrupred inequalities for Kyber512.

Practical Attack Experiments

- Experiment setup
  1. Target: STM32F405 board with ARM Cortex-M4 core
  2. Fault Injection: Instruction skipping via clock glitching
  3. Firmware: Masked implementation based on [BGR+21]



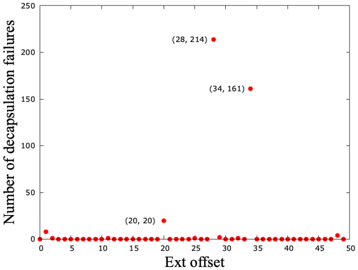
Practical Attack Experiments

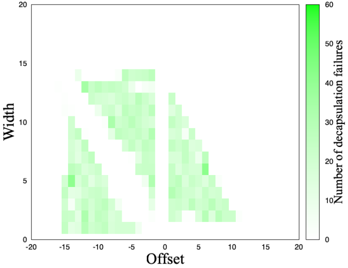
Fault Profiling

1. Fault injection parameters: offset, width, ext_offset, repeat.
2. Scan parameters to find optimal injection timing.

offset	width	ext_offset	repeat
$[-20, 20]$	$[1, 20]$	$[1, 50]$	1

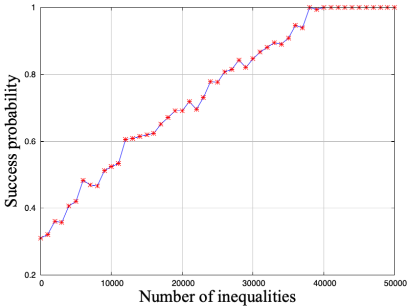
3. Scan (offset, width) pairs to minimize failed fault injections.





Practical Attack Experiments

- Results
  1. With the final fault injection parameters and $\beta = 10$, we collect 50,000 inequalities, showing an error rate of about 6.2%.
  2. Approximately 38,000 inequalities are needed to recover the full secret key, corresponding to 380,000 faulted decapsulations.



Outline

Comparison I

- Comparison under perfect fault injection
  1. This work explores risks introduced by the **non-linear components** in masking implementations.
  2. The collected inequalities are more imbalanced, providing less information.
  3. Consequently, a larger number of inequalities is required, especially for Kyber1024.

	Atatck Target	Type of Inequalities	Security Level	No. Inequalities
This work	Decoder	$\delta \overset{\in}{\underset{\notin}{}} [-192, 192)$	Kyber512	36,000
			Kyber768	54,000
			Kyber1024	4,000,000
[Del22]	Linear Parts	$\delta \gtrless 0$	Kyber512	8,500
			Kyber768	9,400
			Kyber1024	12,000

Comparison II

Comparison in practical attack

1. Both attacks can be performed using clock glitching.
2. Our method achieves higher reliability, resulting in a **higher success rate** with a smaller β.
3. Overall **cost** is **lower**, except for Kyber1024.
4. Unlike methods relying on manipulated ciphertexts (MC), our attack is **harder to defend**.

	Security Level	**No. Inequalities**	β	**Type of Faults**	**MC Req.**
This work	Kyber512 Kyber768 Kyber1024	36,000 54,000 4,000,000	≥ 10	Clock glitch	✗
[Del22]	Kyber512 Kyber768 Kyber1024	8,500 9,400 12,000	> 100	Clock glitch	✔

Comprehensive Comparison

1. Both attacks target the masked decoder.
2. Our method collects inequalities that provide **tighter interval** information, reducing the number of inequalities needed under perfect fault injection.
3. Our method requires a **weaker fault injection**, resulting in significantly **fewer faulted decapsulations** for comparable error rates.

	Type of Inequalities	Security Level	No. Inequalities	β	Type of Faults
This work	$\delta \overset{\in}{\underset{\notin}{}} [-192, 192)$	Kyber512 Kyber768 Kyber1024	36,000 54,000 4,000,000	≥ 10	Clock glitch
[KCS+24]	$\delta \overset{\geq}{\underset{<}{}} -192$	Kyber512	60,000	≥ 180	EM pulse

Reference

Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal.
Masking Kyber: First- and higher-order implementations.
IACR TCHES, 2021(4):173–214, 2021.
https://tches.iacr.org/index.php/TCHES/article/view/9064.

Jeroen Delvaux.
Roulette: A diverse family of feasible fault attacks on masked Kyber.
IACR TCHES, 2022(4):637–660, 2022.

Suparna Kundu, Siddhartha Chowdhury, Sayandeep Saha, Angshuman Karmakar, Debdeep Mukhopadhyay, and Ingrid Verbauwhede.
Carry your fault: A fault propagation attack on side-channel protected LWE-based KEM.
IACR TCHES, 2024(2):844–869, 2024.

Peter Pessl and Lukas Prokop.
Fault attacks on CCA-secure lattice KEMs.
IACR TCHES, 2021(2):37–60, 2021.
https://tches.iacr.org/index.php/TCHES/article/view/8787.

Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma.
Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates.
In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 33–61. Springer, Cham, December 2021.

Thank you for your attention!

Email: wangjian2019@iscas.ac.cn