Number "Not Used" Once - Practical fault attack on pqm4 implementations of NIST candidates

Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, Debdeep Mukhopadhyay

COSADE-2019
5th April 2019

Table of Contents

Table of Contents

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048??

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits ← Millions of physical qubits
- NIST process for standardization of Post-Quantum Cryptography (PQC) is underway.

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- Bristlecone, Google's quantum processor currently works with 72 physical qubits.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits
- NIST process for standardization of Post-Quantum Cryptography (PQC) is underway.
- Started in December 2017, 3-5 years analysis period, followed by 2 years for draft standards.

NIST PQC Call

- Signatures
- Encryption
- Key-establishments (KEMs)
- Selection Criteria:
  - Security
  - Performance
  - Backward compatibility
  - Perfect forward secrecy
  - ...

NIST PQC Call

- Signatures
- Encryption
- Key-establishments (KEMs)
- Selection Criteria:
  - Security
  - Performance
  - Backward compatibility
  - Perfect forward secrecy
  - Resistance to implementation attacks
  - ...

NIST PQC Call

Type	Signatures	KEM/Encryption	Overall
Lattice-based	5	23	28
Code-based	3	17	20
Multivariate	8	2	10
Hash-based	3	0	3
Isogeny-based	0	1	1
Others	2	5	7
Total	21	48	69

NIST PQC Call

Type	Signatures	KEM/Encryption	Overall
Lattice-based	3	9	12
Code-based	0	7	7
Multivariate	4	0	4
Hash-based	2	-	2
Isogeny-based	0	1	1
Others	0	0	0
Total	9	17	26

This Work

- Fault Attack on 4 Lattice-based schemes: **NewHope, Frodo, Kyber, Dilithium**
- Fault Vulnerability: Usage of nonces in the sampling operation.
- Fault Model: **Instruction Skips** on the ARM Cortex-M4.
- Number of faults: *1-5*.
- Nonce-reuse attacks are not new... Well known in the context of ECC.

This Work

- Fault Attack on 4 Lattice-based schemes: **NewHope, Frodo, Kyber, Dilithium**
- Fault Vulnerability: Usage of nonces in the sampling operation.
- Fault Model: **Instruction Skips** on the ARM Cortex-M4.
- Number of faults: *1-5*.
- Nonce-reuse attacks are not new... Well known in the context of ECC.
- Impact:

This Work

- Fault Attack on 4 Lattice-based schemes: **NewHope, Frodo, Kyber, Dilithium**
- Fault Vulnerability: Usage of nonces in the sampling operation.
- Fault Model: **Instruction Skips** on the ARM Cortex-M4.
- Number of faults: *1-5*.
- Nonce-reuse attacks are not new... Well known in the context of ECC.
- Impact:
    - *Key Recovery Attack*

This Work

- Fault Attack on 4 Lattice-based schemes: **NewHope, Frodo, Kyber, Dilithium**
- Fault Vulnerability: Usage of nonces in the sampling operation.
- Fault Model: **Instruction Skips** on the ARM Cortex-M4.
- Number of faults: *1-5*.
- Nonce-reuse attacks are not new... Well known in the context of ECC.
- Impact:
  - *Key Recovery Attack*
  - *Message Recovery Attack in CCA-secure KEM schemes in Man In The Middle (MITM) setting*

Table of Contents

Learning With Errors (LWE) problem

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- Decisional LWE: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^n)$.

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- Decisional LWE: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^n)$.
- Computations over matrices and Vectors were mapped to polynomials in the more efficient variants of LWE such as Ring-LWE (RLWE) and Module-LWE (MLWE).

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- Decisional LWE: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n)$.
- Computations over matrices and Vectors were mapped to polynomials in the more efficient variants of LWE such as Ring-LWE (RLWE) and Module-LWE (MLWE).
- Ring LWE: $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbf{A}, \mathbf{S}, \mathbf{E} \in \mathbf{R}_q$.

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- Decisional LWE: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n)$.
- Computations over matrices and Vectors were mapped to polynomials in the more efficient variants of LWE such as Ring-LWE (RLWE) and Module-LWE (MLWE).
- Ring LWE: $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbf{A}, \mathbf{S}, \mathbf{E} \in \mathbf{R}_q$.
- Module LWE: $\mathbf{R}_q^{k \times l} = (\mathbb{Z}_q[X]/(X^n + 1))^{k \times l}$ with $\mathbf{A} \in \mathbf{R}_q^{k \times \ell}$, $\mathbf{S} \in \mathbf{R}_q^{\ell}$, $\mathbf{E} \in \mathbf{R}_q^{k}$.

Learning With Errors (LWE) problem

- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- Search LWE: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- Decisional LWE: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n)$.
- Computations over matrices and Vectors were mapped to polynomials in the more efficient variants of LWE such as Ring-LWE (RLWE) and Module-LWE (MLWE).
- Ring LWE: $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbf{A}, \mathbf{S}, \mathbf{E} \in \mathbf{R}_q$.
- Module LWE: $\mathbf{R}_q^{k \times l} = (\mathbb{Z}_q[X]/(X^n + 1))^{k \times l}$ with $\mathbf{A} \in \mathbf{R}_q^{k \times \ell}$, $\mathbf{S} \in \mathbf{R}_q^{\ell}$, $\mathbf{E} \in \mathbf{R}_q^{k}$.
- Learning With Rounding (LWR): Error deterministically generated by rounding to a lower modulus.

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme
- Several insecure instantiations of LWE:

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme
- Several insecure instantiations of LWE:
    - Distribution always outputs zero error

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme
- Several insecure instantiations of LWE:
  - Distribution always outputs zero error
  - Distribution always outputs an error in the interval $z + [-\frac{1}{2}, \frac{1}{2})$

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme
- Several insecure instantiations of LWE:
  - Distribution always outputs zero error
  - Distribution always outputs an error in the interval $z + [-\frac{1}{2}, \frac{1}{2})$
  - Sum of a specific set of error co-ordinates is always zero

The Importance of Error

- Error component $\mathbf{E}$ is essential to hardness guarantees
- Without $\mathbf{E}$, LWE instance becomes solvable modular linear equations $T = \mathbf{A} * \mathbf{S}$
- An attack reducing (or bounding) $\mathbf{E}$ could potentially compromise the security of the scheme
- Several insecure instantiations of LWE:
    - Distribution always outputs zero error
    - Distribution always outputs an error in the interval $z + [-\frac{1}{2}, \frac{1}{2})$
    - Sum of a specific set of error co-ordinates is always zero
    - Secret is same as the Error

Table of Contents

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \mathtt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \mathtt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.
- But, we observed...

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.
- But, we observed...
    - $\mathbf{S} = \texttt{Sample}(\sigma, nonce_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma, nonce_{\mathbf{E}})$

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.
- But, we observed...
    - $\mathbf{S} = \texttt{Sample}(\sigma, nonce_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma, nonce_{\mathbf{E}})$
- In need for efficiency, the same seed appended with **one byte of nonce** is used across multiple instances of the *Sample* function.

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.
- But, we observed...
    - $\mathbf{S} = \texttt{Sample}(\sigma, nonce_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma, nonce_{\mathbf{E}})$
- In need for efficiency, the same seed appended with **one byte of nonce** is used across multiple instances of the *Sample* function.
- If this nonce could be faulted to realize *reuse*, then same seed is used to sample both $\mathbf{S}$ and $\mathbf{E}$ resulting in $\mathbf{S} = \mathbf{E}$.

Fault Vulnerability

- Certain amount of randomness required to generate $\mathbf{S}$ and $\mathbf{E}$.
- The secret $\mathbf{S}$ and error $\mathbf{E}$ are sampled from the same distribution and utilize the same functions for sampling.
- $\mathbf{S} = \texttt{Sample}(\sigma_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma_{\mathbf{E}})$
- Ideally, for every fresh instance of `Sample`, one should use a newly generated random seed.
- But, we observed...
    - $\mathbf{S} = \texttt{Sample}(\sigma, nonce_{\mathbf{S}})$, $\mathbf{E} = \texttt{Sample}(\sigma, nonce_{\mathbf{E}})$
- In need for efficiency, the same seed appended with **one byte of nonce** is used across multiple instances of the *Sample* function.
- If this nonce could be faulted to realize *reuse*, then same seed is used to sample both $\mathbf{S}$ and $\mathbf{E}$ resulting in $\mathbf{S} = \mathbf{E}$.

Fault Vulnerability

- Assume a Ring LWE instance

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{E} \in \mathbf{R}_q$$

Fault Vulnerability

- Assume a Ring LWE instance

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{E} \in \mathbf{R}_q$$

- Inject fault such that $\mathbf{E} = \mathbf{S}$.

Fault Vulnerability

- Assume a Ring LWE instance

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{E} \in \mathbf{R}_q$$

- Inject fault such that $\mathbf{E} = \mathbf{S}$.
- Ring-LWE instance is faulted to:

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{S} \in \mathbf{R}_q$$

Fault Vulnerability

- Assume a Ring LWE instance

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{E} \in \mathbf{R}_q$$

- Inject fault such that $\mathbf{E} = \mathbf{S}$.
- Ring-LWE instance is faulted to:

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{S} \in \mathbf{R}_q$$

- Modular linear system of equations with n equations and n unknowns which is trivially solvable.

Fault Vulnerability

- Assume a Ring LWE instance

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{E} \in \mathbf{R}_q$$

- Inject fault such that $\mathbf{E} = \mathbf{S}$.
- Ring-LWE instance is faulted to:

$$\mathbf{T} = \mathbf{A} \times \mathbf{S} + \mathbf{S} \in \mathbf{R}_q$$

- Modular linear system of equations with n equations and n unknowns which is trivially solvable.
- Applies to all variants of LWE (General LWE, Ring-LWE, Module-LWE)

Fault Vulnerability

- These faulty LWE instances can be used to perform key recovery and message recovery attacks.
- Key recovery attacks are performed by faulting the key generation procedure.
- Key recovery attacks applicable to NewHope, Frodo, Kyber and Dilithium.
- Message recovery attacks are performed by faulting the encapsulation procedure.
- Message recovery attacks only applicable over NewHope, Frodo and Kyber KEM schemes.

Table of Contents

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.
- Further, NewHope-CCA KEM is obtained through application of FO transformation on NewHope-CPA KEM.

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.
- Further, NewHope-CCA KEM is obtained through application of FO transformation on NewHope-CPA KEM.
- $\mathbf{S}$ and $\mathbf{E}$ are generated using a `Sample` operation

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.
- Further, NewHope-CCA KEM is obtained through application of FO transformation on NewHope-CPA KEM.
- $\mathbf{S}$ and $\mathbf{E}$ are generated using a `Sample` operation
- `Sample` takes input as a 32-byte seed and 1-byte of *nonce*

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.
- Further, NewHope-CCA KEM is obtained through application of FO transformation on NewHope-CPA KEM.
- $\mathbf{S}$ and $\mathbf{E}$ are generated using a `Sample` operation
- `Sample` takes input as a 32-byte seed and 1-byte of *nonce*
- It uses SHAKE256 (SHA-3 family) as an Extendable Output Function (XOF) to deterministically generate more random bits and subsequently generate $\mathbf{S}$ and $\mathbf{E}$.

NewHope KEM

- NewHope is a suite of KEM (NewHope-CPA/CCA-KEM)
- Based on RLWE problem
- NewHope-CPA KEM is derived from the NewHope-CPA Public Key Encryption (PKE) scheme.
- Further, NewHope-CCA KEM is obtained through application of FO transformation on NewHope-CPA KEM.
- $\mathbf{S}$ and $\mathbf{E}$ are generated using a `Sample` operation
- `Sample` takes input as a 32-byte seed and 1-byte of *nonce*
- It uses SHAKE256 (SHA-3 family) as an Extendable Output Function (XOF) to deterministically generate more random bits and subsequently generate $\mathbf{S}$ and $\mathbf{E}$.
- In NIST submission, designers use nonce=(0,1).

NEWHOPE CPA-PKE

procedure NEWHOPE.CPAPKE.GEN()

$\vdots$

$\hat{\mathbf{a}} \leftarrow \texttt{GenA}(publicseed)$

$\mathbf{s} \leftarrow \texttt{PolyBitRev}(\texttt{Sample}(noiseseed, 0))$

$\hat{\mathbf{s}} = \texttt{NTT}(\mathbf{s})$

$\mathbf{e} \leftarrow \texttt{PolyBitRev}(\texttt{Sample}(noiseseed, 1))$

$\hat{\mathbf{e}} = \texttt{NTT}(\mathbf{e})$

$\hat{\mathbf{b}} = \hat{\mathbf{a}} * \hat{\mathbf{s}} + \hat{\mathbf{e}}$

Return $(pk = \texttt{EncodePK}(\hat{\mathbf{b}}, publicseed), sk = \texttt{EncodePolynomial}(\mathbf{s}))$

end procedure

NEWHOPE CPA-PKE

```
procedure NEWHOPE.CPAPKE.GEN()
    ⋮
    â ← GenA(publicseed)
    s ← PolyBitRev(Sample(noiseseed, 0→ R))
    ŝ = NTT(s)
    e ← PolyBitRev(Sample(noiseseed, 1→ R))
    ê = NTT(e)
    b̂ = â ∗ ŝ + ê
    Return
(pk = EncodePK(b̂, publicseed), sk = EncodePolynomial(s))
end procedure
```

NEWHOPE CPA-PKE

procedure NEWHOPE.CPAPKE.GEN()

$\vdots$

$\hat{\mathbf{a}} \leftarrow \texttt{GenA}(publicseed)$

$\mathbf{s} \leftarrow \texttt{PolyBitRev}(\texttt{Sample}(noiseseed, 0 \rightarrow R))$

$\hat{\mathbf{s}} = \texttt{NTT}(\mathbf{s})$

$\mathbf{e} \leftarrow \texttt{PolyBitRev}(\texttt{Sample}(noiseseed, 1 \rightarrow R))$

$\hat{\mathbf{e}} = \texttt{NTT}(\mathbf{e})$

$\hat{\mathbf{b}} = \hat{\mathbf{a}} * \hat{\mathbf{s}} + \hat{\mathbf{e}}$

Return $(pk = \texttt{EncodePK}(\hat{\mathbf{b}}, publicseed), sk = \texttt{EncodePolynomial}(\mathbf{s}))$

end procedure

Frodo KEM

- Frodo, similar to NewHope is a suite of KEM (NewHope-CPA/CCA-KEM) based on the General LWE problem.
- We identify the same vulnerable usage of nonce for sampling $\mathbf{S}$ and $\mathbf{E}$.

Frodo CPA-PKE

procedure `FRODO.CPAPKE.GEN()`

$\quad seed_{\mathbf{A}} \leftarrow U(\{0,1\}^{len_A})$

$\quad \mathbf{A} \leftarrow \text{Frodo.Gen}(seed_{\mathbf{A}}) \in \mathbb{Z}_q^{n \times n}$

$\quad seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_E})$

$\quad \mathbf{S} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 1) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\quad \mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 2) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\quad \mathbf{B} = \mathbf{A} \times \mathbf{S} + \mathbf{E}$

$\quad$ Public key $pk \leftarrow (seed_{\mathbf{A}}, \mathbf{B})$ and Secret key $sk \leftarrow \mathbf{S}$

end procedure

Frodo CPA-PKE

procedure FRODO.CPAPKE.GEN()

$\quad seed_{\mathbf{A}} \leftarrow U(\{0,1\}^{len_A})$

$\quad \mathbf{A} \leftarrow \text{Frodo.Gen}(seed_{\mathbf{A}}) \in \mathbb{Z}_q^{n \times n}$

$\quad seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_E})$

$\quad \mathbf{S} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 1 \rightarrow R) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\quad \mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 2 \rightarrow R) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\quad \mathbf{B} = \mathbf{A} \times \mathbf{S} + \mathbf{E}$

$\quad$ Public key $pk \leftarrow (seed_{\mathbf{A}}, \mathbf{B})$ and Secret key $sk \leftarrow \mathbf{S}$

end procedure

Frodo CPA-PKE

procedure FRODO.CPAPKE.GEN()

$seed_{\mathbf{A}} \leftarrow U(\{0,1\}^{len_A})$

$\mathbf{A} \leftarrow \text{Frodo.Gen}(seed_{\mathbf{A}}) \in \mathbb{Z}_q^{n \times n}$

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_E})$

$\mathbf{S} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 1 \rightarrow R) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 2 \rightarrow R) \in \mathbb{Z}_q^{n \times \bar{n}}$

$\mathbf{B} = \mathbf{A} \times \mathbf{S} + \mathbf{E}$

Public key $pk \leftarrow (seed_{\mathbf{A}}, \mathbf{B})$ and Secret key $sk \leftarrow \mathbf{S}$

end procedure

Kyber KEM

- Kyber is a suite of KEM (NewHope-CPA/CCA-KEM) based on the MLWE problem
- $\mathbf{S} \in R_q^k$ and $\mathbf{E} \in \mathbf{R}_q^\ell$ are sampled from a Centered Binomial distribution.
- Same seeds appended with fixed nonces are yet again used in sampling $\mathbf{S}$ and $\mathbf{E}$.

Kyber KEM

- Kyber is a suite of KEM (NewHope-CPA/CCA-KEM) based on the MLWE problem
- $\mathbf{S} \in R_q^k$ and $\mathbf{E} \in \mathbf{R}_q^\ell$ are sampled from a Centered Binomial distribution.
- Same seeds appended with fixed nonces are yet again used in sampling $\mathbf{S}$ and $\mathbf{E}$.
- In NIST submission, designers use nonce=(0 to k-1) for $\mathbf{S}$ and nonce=(k to 2k-1) for $\mathbf{E}$.

Kyber CPA-PKE

procedure KYBER.CPAPKE.GEN()
    $d \leftarrow \{0,1\}^{256}$, $(\rho, \sigma) := G(d)$, $N := 0$
For i from 0 to $k-1$
For j from 0 to $k-1$
    $\mathbf{a}[i][j] \leftarrow \text{Parse}(\text{XOF}(\rho || j || i))$
EndFor
EndFor
For i from 0 to $k-1$
    $\mathbf{s}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N))$
    $N := N + 1$
EndFor
For i from 0 to $k-1$
    $\mathbf{e}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N))$
    $N := N + 1$
EndFor
    $\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$
    $\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{a}} * \hat{\mathbf{s}}) + \mathbf{e}$
    $pk := (\text{Encode}_{d_t}(\text{Compress}_q(\mathbf{t}, d_t)) || \rho)$
    Secret Key $:= \text{Encode}_{13}(\hat{\mathbf{s}} \; mod^{+} q)$
    Return (Public Key,Secret Key)
end procedure

Kyber CPA-PKE

procedure KYBER.CPAPKE.GEN()
    $d \leftarrow \{0,1\}^{256}$, $(\rho, \sigma) := G(d)$, $N := 0$
For i from 0 to $k-1$
For j from 0 to $k-1$
    $\mathbf{a}[i][j] \leftarrow \text{Parse}(\text{XOF}(\rho||j||i))$
EndFor
EndFor
For i from 0 to $k-1$
    $\mathbf{s}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
    $N := N + 1$
EndFor
For i from 0 to $k-1$
    $\mathbf{e}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
    $N := N + 1$
EndFor
    $\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$
    $\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{a}} * \hat{\mathbf{s}}) + \mathbf{e}$
    $pk := (\text{Encode}_{d_t}(\text{Compress}_q(\mathbf{t}, d_t))||\rho)$
    Secret Key $:= \text{Encode}_{13}(\hat{\mathbf{s}}\ mod^+ q)$
    Return (Public Key,Secret Key)
end procedure

Kyber CPA-PKE

procedure KYBER.CPAPKE.GEN()
  $d \leftarrow \{0,1\}^{256}$, $(\rho, \sigma) := G(d)$, $N := 0$
For i from 0 to $k-1$
For j from 0 to $k-1$
  $\mathbf{a}[i][j] \leftarrow \text{Parse}(\text{XOF}(\rho||j||i))$
EndFor
EndFor
For i from 0 to $k-1$
  $\mathbf{s}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
  $N := N + 1$
EndFor
For i from 0 to $k-1$
  $\mathbf{e}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
  $N := N + 1$
EndFor
  $\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$
  $\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{a}} * \hat{\mathbf{s}}) + \mathbf{e}$
  $pk := (\text{Encode}_{d_t}(\text{Compress}_q(\mathbf{t}, d_t))||\rho)$
  Secret Key $:= \text{Encode}_{13}(\hat{\mathbf{s}}\ mod^{+} q)$
  Return (Public Key,Secret Key)
end procedure

Kyber CPA-PKE

procedure KYBER.CPAPKE.GEN()
$d \leftarrow \{0,1\}^{256}$, $(\rho, \sigma) := G(d)$, $N := 0$
For i from 0 to $k-1$
For j from 0 to $k-1$
$\mathbf{a}[i][j] \leftarrow \text{Parse}(\text{XOF}(\rho||j||i))$
EndFor
EndFor
For i from 0 to $k-1$
$\mathbf{s}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
$N := N + 1$
EndFor
For i from 0 to $k-1$
$\mathbf{e}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(\sigma, N \rightarrow R))$
$N := N + 1$
EndFor
$\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$
$\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{a}} * \hat{\mathbf{s}}) + \mathbf{e}$
Public Key $:= (\text{Encode}_{d_t}(\text{Compress}_q(\mathbf{t}, d_t))||\rho)$ **** Adds more error
Secret Key $:= \text{Encode}_{13}(\hat{\mathbf{s}}\ mod^+ q)$
Return (Public Key,Secret Key)
end procedure

Key Recovery Attack on Kyber

- The Compress function rounds each coefficient to a lower modulus thereby inherently introducing additional deterministic error.
- Though the induced fault nullified the error in the LWE instance, the LWR hardness might stil not be possible to break.

Key Recovery Attack on Kyber

- The Compress function rounds each coefficient to a lower modulus thereby inherently introducing additional deterministic error.
- Though the induced fault nullified the error in the LWE instance, the LWR hardness might stil not be possible to break.
- The authors have only considered rounding for efficiency and not for security.

Key Recovery Attack on `Kyber`

- The Compress function rounds each coefficient to a lower modulus thereby inherently introducing additional deterministic error.
- Though the induced fault nullified the error in the LWE instance, the LWR hardness might stil not be possible to break.
- The authors have only considered rounding for efficiency and not for security.
- The authors state that "we believe that the compression technique adds some security", but it has not been quantified.

Key Recovery Attack on Kyber

- The Compress function rounds each coefficient to a lower modulus thereby inherently introducing additional deterministic error.
- Though the induced fault nullified the error in the LWE instance, the LWR hardness might stil not be possible to break.
- The authors have only considered rounding for efficiency and not for security.
- The authors state that "we believe that the compression technique adds some security", but it has not been quantified.
- Thus, our fault does not result in direct key recovery attack, but brings down the hardness to solving the corresponding LWR problem.

Dilithium Signature Scheme

- Dilithium is a Fiat-Shamir Abort-based lattice signature scheme.
- Indistinguishability of the Public key is based on the MLWE problem.
- Here again, nonces appended with domain separators are used to sample $\mathbf{S} \in \mathbf{R}_q^{\ell}$ and $\mathbf{E} \in \mathbf{R}_q^{k}$.

Dilithium Signature Scheme

procedure DILITHIUM.KEYGEN()
  $\rho, \rho' \leftarrow \{0,1\}^{256}$, $K \leftarrow \{0,1\}^{256}$, $N := 0$
For i from 0 to $\ell - 1$
  $\mathbf{s}_1[i] := Sample(PRF(\rho', N))$
  $N := N + 1$
EndFor
For i from 0 to $k - 1$
  $\mathbf{s}_2[i] := Sample(PRF(\rho', N))$
  $N := N + 1$
EndFor $\mathbf{A} \sim R_q^{k \times \ell} :=$ ExpandA(ρ)
  Compute $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$
  Compute $\mathbf{t}_1 :=$ Power2Round$_q(\mathbf{t}, d)$
  $tr \in \{0,1\}^{384} :=$ CRH$(\rho || \mathbf{t}_1)$
  Return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
end procedure

Dilithium Signature Scheme

procedure DILITHIUM.KEYGEN()
    $\rho, \rho' \leftarrow \{0,1\}^{256}$, $K \leftarrow \{0,1\}^{256}$, $N := 0$
For i from 0 to $\ell - 1$
    $\mathbf{s}_1[i] := Sample(PRF(\rho', N \rightarrow R))$
    $N := N + 1$
EndFor
For i from 0 to $k - 1$
    $\mathbf{s}_2[i] := Sample(PRF(\rho', N \rightarrow R))$
    $N := N + 1$
EndFor $\mathbf{A} \sim R_q^{k \times \ell} :=$ ExpandA(ρ)
    Compute $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$
    Compute $\mathbf{t}_1 :=$ Power2Round$_q(\mathbf{t}, d)$
    $tr \in \{0,1\}^{384} :=$ CRH$(\rho || \mathbf{t}_1)$
    Return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
end procedure

Dilithium Signature Scheme

procedure DILITHIUM.KEYGEN()

$\rho, \rho' \leftarrow \{0,1\}^{256}$, $K \leftarrow \{0,1\}^{256}$, $N := 0$

For i from 0 to $\ell - 1$

$\mathbf{s}_1[i] := Sample(PRF(\rho', N \rightarrow R))$

$N := N + 1$

EndFor

For i from 0 to $k - 1$

$\mathbf{s}_2[i] := Sample(PRF(\rho', N \rightarrow R))$

$N := N + 1$

EndFor $\mathbf{A} \sim R_q^{k \times \ell} :=$ ExpandA(ρ)

Compute $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$

Compute $\mathbf{t}_1 :=$ Power2Round$_q(\mathbf{t}, d)$

$tr \in \{0,1\}^{384} :=$ CRH$(\rho || \mathbf{t}_1)$

Return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

end procedure

Dilithium Signature Scheme

procedure DILITHIUM.KEYGEN()
    $\rho, \rho' \leftarrow \{0,1\}^{256}$, $K \leftarrow \{0,1\}^{256}$, $N := 0$
For i from 0 to $\ell - 1$
    $\mathbf{s}_1[i] := Sample(PRF(\rho', N \rightarrow R))$
    $N := N + 1$
EndFor
For i from 0 to $k - 1$
    $\mathbf{s}_2[i] := Sample(PRF(\rho', N \rightarrow R))$
    $N := N + 1$
EndFor $\mathbf{A} \sim R_q^{k \times \ell} :=$ ExpandA(ρ)
    Compute $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$
    Compute $\mathbf{t}_1 :=$ Power2Round$_q(\mathbf{t}, d)$ ***** Only the top d bits of $\mathbf{t}$
    $tr \in \{0,1\}^{384} :=$ CRH$(\rho || \mathbf{t}_1)$
    Return $pk = (\rho, \mathbf{t}_1)$, $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
end procedure

Key Recovery Attack on `Dilithium`

- Only the higher order bits of the LWE instance $\mathbf{t}$ are declared as the public key.

Key Recovery Attack on Dilithium

- Only the higher order bits of the LWE instance $\mathbf{t}$ are declared as the public key.
- Some rounding error is introduced on top of the LWE instance $\mathbf{t}$.

Key Recovery Attack on `Dilithium`

- Only the higher order bits of the LWE instance $\mathbf{t}$ are declared as the public key.
- Some rounding error is introduced on top of the LWE instance $\mathbf{t}$.
- Security Analysis of Dilithium assumes that the whole of $\mathbf{t}$ is known to the adversary. The original LWE instance $\mathbf{t}$ can be derived just through observation of a large number of signatures.

Key Recovery Attack on `Dilithium`

- Only the higher order bits of the LWE instance $\mathbf{t}$ are declared as the public key.
- Some rounding error is introduced on top of the LWE instance $\mathbf{t}$.
- Security Analysis of Dilithium assumes that the whole of $\mathbf{t}$ is known to the adversary. The original LWE instance $\mathbf{t}$ can be derived just through observation of a large number of signatures.
- If the whole of $\mathbf{t}$ can be derived by the adversary, our induced faults results in a key recovery attack.

Table of Contents

NEWHOPE CPA-PKE

procedure
$\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0, \ldots, 255\}^{7.n/4+32}, \mu \in \{0, \ldots, 255\}^{32}, coin \in \{0, \ldots, 255\}^{32})$

$\quad\vdots$
$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0))$
$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1))$
$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$
$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$
$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$
$\quad \mathbf{v} = \texttt{Encode}(\mu)$
$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$
$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$
$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad\acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(\mathit{Sample}(coin, 0 \rightarrow R))$

$\quad\acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(\mathit{Sample}(coin, 1 \rightarrow R))$

$\quad\acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad\acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad\hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad\mathbf{v} = \texttt{Encode}(\mu)$

$\quad\acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad\mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure NEWHOPE.CPAPKE.ENC($pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32}$)

$\quad \vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

NEWHOPE CPA-PKE

procedure $\texttt{NEWHOPE.CPAPKE.ENC}(pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32})$

$\quad\vdots$

$\quad \acute{\mathbf{s}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 0 \rightarrow R))$

$\quad \acute{\mathbf{e}} \leftarrow \texttt{PolyBitRev}(Sample(coin, 1 \rightarrow R))$

$\quad \acute{\acute{\mathbf{e}}} \leftarrow \texttt{Sample}(coin, 2)$

$\quad \acute{\mathbf{t}} = \texttt{NTT}(\acute{\mathbf{s}})$

$\quad \hat{\mathbf{u}} = \hat{\mathbf{a}} * \hat{\mathbf{t}} + \texttt{NTT}(\acute{\mathbf{e}})$

$\quad \mathbf{v} = \texttt{Encode}(\mu)$

$\quad \acute{\mathbf{v}} = \texttt{NTT}^{-1}(\hat{\mathbf{b}} * \hat{\mathbf{t}}) + \acute{\acute{\mathbf{e}}} + \mathbf{v}$

$\quad \mathbf{h} = \texttt{Compress}(\acute{\mathbf{v}})$

$\quad$ Return $c = \texttt{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$

end procedure

FRODO CPA-PKE

procedure `FRODO.CPAPKE.ENC()`

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n \times \bar{n}}$

Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

FRODO CPA-PKE

procedure FRODO.CPAPKE.ENC()

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n \times \bar{n}}$

Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

FRODO CPA-PKE

procedure FRODO.CPAPKE.ENC()

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n \times \bar{n}}$

Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

FRODO CPA-PKE

procedure `FRODO.CPAPKE.ENC()`

$\quad seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\quad \acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4 \rightarrow R) \in \mathbb{Z}_q^{\bar{m}\times n}$

$\quad \acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5 \rightarrow R) \in \mathbb{Z}_q^{\bar{m}\times n}$

$\quad \acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n\times \bar{n}}$

$\quad$ Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

$\quad$ Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

$\quad$ Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

FRODO CPA-PKE

procedure FRODO.CPAPKE.ENC()

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5 \rightarrow R) \in \mathbb{Z}_q^{\bar{m} \times n}$

$\acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n \times \bar{n}}$

Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

FRODO CPA-PKE

procedure FRODO.CPAPKE.ENC()

$seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_{\mathbf{E}}})$

$\acute{\mathbf{S}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 4 \rightarrow R) \in \mathbb{Z}_q^{\bar{m}\times n}$

$\acute{\mathbf{E}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 5 \rightarrow R) \in \mathbb{Z}_q^{\bar{m}\times n}$

$\acute{\acute{\mathbf{E}}} \leftarrow \text{Frodo.SampleMatrix}(seed_{\mathbf{E}}, 6) \in \mathbb{Z}_q^{n\times \bar{n}}$

Compute $\acute{\mathbf{B}} = \acute{\mathbf{S}} \times \mathbf{A} + \acute{\mathbf{E}}$

Compute $\mathbf{V} = \acute{\mathbf{S}} \times \mathbf{B} + \acute{\acute{\mathbf{E}}} + \text{Frodo.Encode}(\mu)$

Ciphertext $\mathbf{C} \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\acute{\mathbf{B}}, \mathbf{V})$

end procedure

KYBER CPA-PKE

procedure KYBER.CPAPKE.ENC($pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$, $m \in \mathcal{B}^{32}$, $r \in \mathcal{B}^{32}$)

$\quad N = 0$

For i from 0 to $k-1$

$\quad \mathbf{r}[i] \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$

$\quad N := N + 1$

EndFor

For i from 0 to $k-1$

$\quad \mathbf{e}_1 \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$

$\quad N := N + 1$

EndFor

For i from 0 to $k-1$ $\mathbf{e}_2 \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$

EndFor

$\quad \hat{\mathbf{r}} = \mathsf{NTT}(\mathbf{r})$

$\quad \mathbf{u} = \mathsf{NTT}^{-1}(\hat{a}^T * \hat{\mathbf{r}}) + \mathbf{e}_1$

$\quad \mathbf{v} = \mathsf{NTT}^{-1}(\hat{t}^T * \hat{\mathbf{r}}) + \mathbf{e}_2 + \mathsf{Decode}_1(\mathsf{Decompose}_q(m, 1))$

$\quad \mathbf{c}_1 = \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$

$\quad \mathbf{c}_2 = \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(\mathbf{v}, d_v))$

$\quad \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$

end procedure

KYBER CPA-PKE

procedure $\texttt{KYBER.CPAPKE.ENC}(pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}, m \in \mathcal{B}^{32}, r \in \mathcal{B}^{32})$
$N = 0$
For i from 0 to $k-1$
$\mathbf{r}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$
$N := N + 1$
EndFor
For i from 0 to $k-1$
$\mathbf{e}_1 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$
$N := N + 1$
EndFor
For i from 0 to $k-1$ $\mathbf{e}_2 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N))$
EndFor
$\hat{\mathbf{r}} = \text{NTT}(\mathbf{r})$
$\mathbf{u} = \text{NTT}^{-1}(\hat{a}^T * \hat{\mathbf{r}}) + \mathbf{e}_1$
$\mathbf{v} = \text{NTT}^{-1}(\hat{t}^T * \hat{\mathbf{r}}) + \mathbf{e}_2 + \text{Decode}_1(\text{Decompose}_q(m, 1))$
$\mathbf{c}_1 = \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$
$\mathbf{c}_2 = \text{Encode}_{d_v}(\text{Compress}_q(\mathbf{v}, d_v))$
$\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$
end procedure

KYBER CPA-PKE

procedure $\texttt{KYBER.CPAPKE.ENC}(pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32},\ m \in \mathcal{B}^{32},\ r \in \mathcal{B}^{32})$

  $N = 0$

For i from 0 to $k-1$

  $\mathbf{r}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$

  $N := N + 1$

EndFor

For i from 0 to $k-1$

  $\mathbf{e}_1 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$

  $N := N + 1$

EndFor

For i from 0 to $k-1$ $\mathbf{e}_2 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N))$

EndFor

  $\hat{\mathbf{r}} = \text{NTT}(\mathbf{r})$

  $\mathbf{u} = \text{NTT}^{-1}(\hat{a}^T * \hat{\mathbf{r}}) + \mathbf{e}_1$

  $\mathbf{v} = \text{NTT}^{-1}(\hat{t}^T * \hat{\mathbf{r}}) + \mathbf{e}_2 + \text{Decode}_1(\text{Decompose}_q(m, 1))$

  $\mathbf{c}_1 = \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$

  $\mathbf{c}_2 = \text{Encode}_{d_v}(\text{Compress}_q(\mathbf{v}, d_v))$

  $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$

end procedure

KYBER CPA-PKE

procedure $\texttt{KYBER.CPAPKE.ENC}(pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}, m \in \mathcal{B}^{32}, r \in \mathcal{B}^{32})$

$N = 0$

For i from 0 to $k-1$

$\mathbf{r}[i] \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$

$N := N + 1$

EndFor

For i from 0 to $k-1$

$\mathbf{e}_1 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N \rightarrow R))$

$N := N + 1$

EndFor

For i from 0 to $k-1$ $\mathbf{e}_2 \leftarrow \text{CBD}_\eta(\text{PRF}(r, N))$

EndFor

$\hat{\mathbf{r}} = \text{NTT}(\mathbf{r})$

$\mathbf{u} = \text{NTT}^{-1}(\hat{a}^T * \hat{\mathbf{r}}) + \mathbf{e}_1$

$\mathbf{v} = \text{NTT}^{-1}(\hat{t}^T * \hat{\mathbf{r}}) + \mathbf{e}_2 + \text{Decode}_1(\text{Decompose}_q(m, 1))$

$\mathbf{c}_1 = \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$ **** Adds more error

$\mathbf{c}_2 = \text{Encode}_{d_v}(\text{Compress}_q(\mathbf{v}, d_v))$

$\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$

end procedure

Translating Message Recovery Attack to CCA-KEM schemes

- CPA-secure PKE is transformed to CCA-secure KEM using the Quantum-Fujisaki Okamoto transformation.
- A **re-encapsulation** is performed in the decapsulation procedure to check for the validity of ciphertexts.

Translating Message Recovery Attack to CCA-KEM schemes

- CPA-secure PKE is transformed to CCA-secure KEM using the Quantum-Fujisaki Okamoto transformation.
- A **re-encapsulation** is performed in the decapsulation procedure to check for the validity of ciphertexts.
- Thus, faults injected into the encapsulation procedure are detected during decapsulation.

Translating Message Recovery Attack to CCA-KEM schemes

- CPA-secure PKE is transformed to CCA-secure KEM using the Quantum-Fujisaki Okamoto transformation.
- A **re-encapsulation** is performed in the decapsulation procedure to check for the validity of ciphertexts.
- Thus, faults injected into the encapsulation procedure are detected during decapsulation.
- How do we bypass this?

Translating Message Recovery Attack to CCA-KEM schemes

- CPA-secure PKE is transformed to CCA-secure KEM using the Quantum-Fujisaki Okamoto transformation.
- A **re-encapsulation** is performed in the decapsulation procedure to check for the validity of ciphertexts.
- Thus, faults injected into the encapsulation procedure are detected during decapsulation.
- How do we bypass this?
- We observe that a fault attacker in a Man-In-The-Middle (MITM) setting can still mount the attack without being detected during decapsulation.

Message Recovery Attack over CCA-KEM schemes

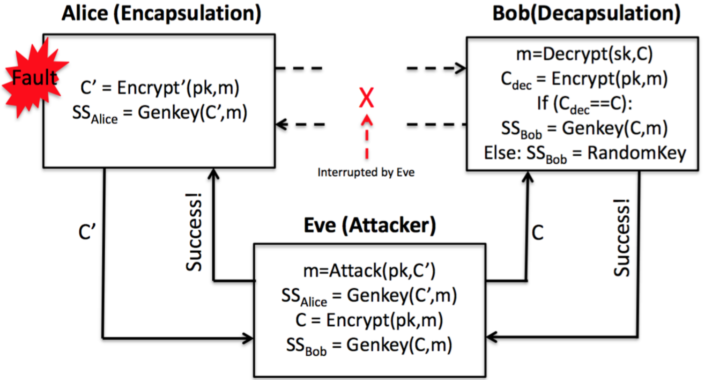


Figure: Fault assisted MITM attack on CCA Secure KEM scheme

Table of Contents

Experimental Validation on ARM Cortex-M4

- We target reference implementations from the *pqm4* benchmarking framework for PQC candidates on the ARM Cortex-M4 microcontroller.
- Implementations were ported to the STM32F4DISCOVERY board (DUT) housing the STM32F407 microcontroller.
- Clock Frequency: 24 MHz.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

- For all the call instances to this XOF function, all the elements of the array A are the same except the nonce value.

Analysis of implementation for Fault Vulnerability

- We target the usage (not generation) of nonce in all reference implementations.
- The seed to the `Sample` function along with the nonce is input as an array A to an XOF function.
- The nonce is stored as the last element of the array.

- For all the call instances to this XOF function, all the elements of the array A are the same except the nonce value.
- If this *nonce-store* to the array is skipped, we are essentially using the same randomness to sample both $\mathbf{S}$ and $\mathbf{E}$.

Analysis of implementation for Fault Vulnerability

```
ldr     r3,[r5,#28]
stmia   r4!,{r0,r1,r2,r3}
strb.w  r7,[r6,#-132]!
movs    r1,#1
mov     r0,r6
```

(a) Target operation in NewHope

```
movs    r1,#1
add     r0,sp,#52
strb.w  r9,[r6,#32]
movs    r2,#33
movs    r3,#0
```

(b) Target operation in Kyber

```
lsrs    r2,r7,#8
ldr     r3,[pc,#264]
strb.w  r2,[sp,#7]
movw    r2,#4097
mov     r1,sp
```

(c) Target operation in Frodo

```
movs    r1,#128
ldr     r0,[pc,#208]
strb.w  r7,[sp,#44]
add     r1,sp,#12
add     r0,sp,#48
```

(d) Target operation in Dilithium

Experimental Setup

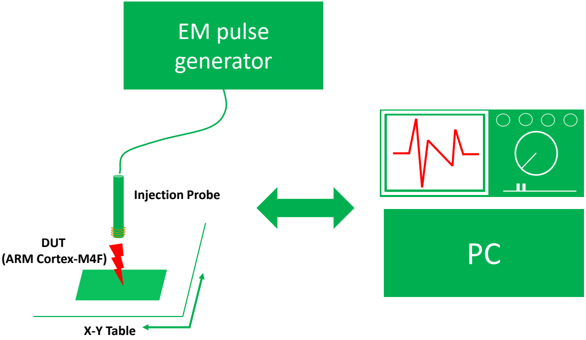


Figure: Description of our EMFI setup

Experimental Setup

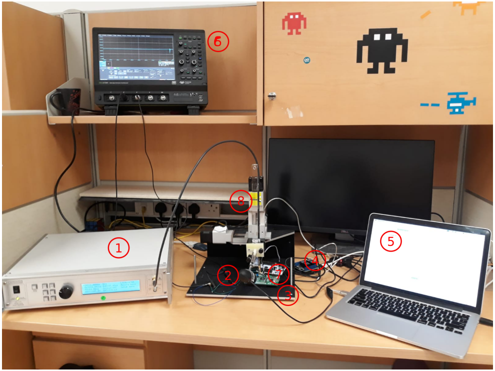
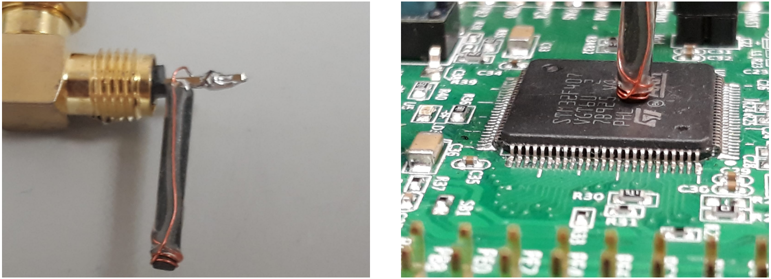


Figure: (1) EM Pulse Generator (2) USB-Microscope (3) STM32M4F Discovery Board (DUT) (4) Arudino based Relay Shield (5) Controller Laptop (6) Oscilloscope (7) EM Pulse Injector (8) XYZ Motorized Table

Experimental Setup



(a) (b)

Figure: (a) Hand-made probe used for our EMFI setup (b) Probe placed over the DUT

Results on ARM Cortex-M4

- **Required Fault:** Skip the store instruction to a particular memory location.
- We profiled the ARM chip using a sample load and store program to find a "sweet spot" to skip the store to a particular memory location.

Results on ARM Cortex-M4

- **Required Fault:** Skip the store instruction to a particular memory location.
- We profiled the ARM chip using a sample load and store program to find a "sweet spot" to skip the store to a particular memory location.
- Fault sensitive region is the area near the ARM logo of the STM32M4F07 microcontroller.

Results on ARM Cortex-M4

- **Required Fault:** Skip the store instruction to a particular memory location.
- We profiled the ARM chip using a sample load and store program to find a ”sweet spot” to skip the store to a particular memory location.
- Fault sensitive region is the area near the ARM logo of the STM32M4F07 microcontroller.
- Fault repeatability is (almost) 100% at the identified location for a specific set of voltage pulse parameters.

Results on ARM Cortex-M4

- **Required Fault:** Skip the store instruction to a particular memory location.
- We profiled the ARM chip using a sample load and store program to find a "sweet spot" to skip the store to a particular memory location.
- Fault sensitive region is the area near the ARM logo of the STM32M4F07 microcontroller.
- Fault repeatability is (almost) 100% at the identified location for a specific set of voltage pulse parameters.
- Voltage:150V-200V, Pulse Width = 12nsec, Rise-Time = 2 nsec.

Results on ARM Cortex-M4

- **Required Fault:** Skip the store instruction to a particular memory location.
- We profiled the ARM chip using a sample load and store program to find a "sweet spot" to skip the store to a particular memory location.
- Fault sensitive region is the area near the ARM logo of the STM32M4F07 microcontroller.
- Fault repeatability is (almost) 100% at the identified location for a specific set of voltage pulse parameters.
- Voltage:150V-200V, Pulse Width = 12nsec, Rise-Time = 2 nsec.
- Faults were synchronized with the target operation using an internally generated trigger.

Fault Complexity

Attack Objective	Fault Complexity			
	NEWHOPE		FRODO	
	NEWHOPE512	NEWHOPE1024	Frodo-640	Frodo-976
Key Recovery	1	1	1	1
Message Recovery	1	1	1	1

	KYBER			DILITHIUM			
	KYBER512	KYBER768	KYBER1024	Weak	Med.	Rec.	High
Key Recovery	2	3	4	2	3	4	5
Message Recovery	2	3	4	-	-	-	-

Fault Complexity

Attack Objective	Fault Complexity			
	NEWHOPE		FRODO	
	NEWHOPE512	NEWHOPE1024	Frodo-640	Frodo-976
Key Recovery	1	1	1	1
Message Recovery	1	1	1	1

	KYBER			DILITHIUM			
	KYBER512	KYBER768	KYBER1024	Weak	Med.	Rec.	High
Key Recovery	2	3	4	2	3	4	5
Message Recovery	2	3	4	-	-	-	-

- Security of `Kyber` is weakened because the induced fault has removed the hardness from the LWE problem.

Fault Complexity

Attack Objective	Fault Complexity			
	NEWHOPE		FRODO	
	NEWHOPE512	NEWHOPE1024	Frodo-640	Frodo-976
Key Recovery	1	1	1	1
Message Recovery	1	1	1	1

	KYBER			DILITHIUM			
	KYBER512	KYBER768	KYBER1024	Weak	Med.	Rec.	High
Key Recovery	2	3	4	2	3	4	5
Message Recovery	2	3	4	-	-	-	-

- Security of Kyber is weakened because the induced fault has removed the hardness from the LWE problem.
- If enough number of signatures corresponding to the same public-private key pair can be observed, then it can lead to a successful key recovery attack on Dilithium.

Table of Contents

Countermeasures and Future Directions

- Usage of separate seeds for $\mathbf{S}$ and $\mathbf{E}$
- `Frodo` has updated its specifications as part of its second round submission by using separate seeds for $\mathbf{S}$ and $\mathbf{E}$.
- Synchronization of faults with vulnerable operations.
- Study on weakened LWE instances in Kyber and Dilithium.

Table of Contents

Conclusion

- We identified fault-vulnerabilities due to usage of nonces in multiple LWE-based lattice schemes.
- We proposed key recovery attacks over `NewHope`, `Frodo`, `Kyber` and `Dilithium` and message recovery attacks over `NewHope`, `Frodo` and `Kyber` KEM schemes.
- Practical Validation of our attack through EMFI on implementations from *pqm4* library on the ARM Cortex-M4 microcontroller.
- We hope that nonces either be avoided or be used more carefully in the future.

Thank you!

Any questions?