Saber Post-Quantum Key Encapsulation Mechanism (KEM): Evaluating Performance in Mobile Devices and Suggesting Some Improvements / Evaluating Kyber in post-quantum KEM in a mobile application
Topics

• Saber/Kyber Testing Flow
• Saber Performance Tests Data
  • x64 versus ARM Architectures
• Kyber Performance Tests Data
  • x64 versus ARM Architectures
Saber/Kyber Test Flow

1. **Performance Testing Code in Java**
2. **JNI**
3. **Compile**
4. **Testing Code Classes**
5. **Run**
6. **X64**
7. **Profiling Data**

- **Saber/Kyber Code in C**
- **JNI Header C Code**
- **Performance Testing Code in C**
- **kem.so**
- **Compile Gcc/NDK**
- **Run**
- **ARMv8**
- **30 Executions of 1 Round for Round 2 Code**
- **30 Executions of 1000 Rounds for Round 3 Code**
How were the tests done?

- Used standard version of both Saber/Kyber.
- Tests Characteristics:
  - Input: Key Session Object (128 bytes).
  - Output: Profiling Data
  - Code Sequence:
    - Call “indcpa_kem_keypair (byte[] pubKey, byte[] privKey)”.
    - Call “indcpa_kem_enc (byte[] message, byte[] pubKey)”.
    - Call “indcpa_kem_dec (byte[] encData, byte[] privKey)”.
- Padding was necessary when data was not multiple of block size.
Algorithms Versions Evaluated - NIST Round 2 and 3

- **Kyber1024**
  - NIST security level: 5¹
  - sk: 3168
  - pk: 1568
  - ct: 1568

- **FireSaber**
  - NIST security level: 5¹
  - sk: 1664
  - pk: 1312
  - ct: 1472

¹ “Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g. AES 256).” (NIST, 2017).
Saber Test Devices

- Mobile Device ARMv8
  - Android 10
  - RAM: 8GB
  - Octa-core (2x2.73 GHz Mongoose M5 + 2x2.60 GHz Cortex-A76 + 4x2.0 GHz Cortex-A55)

- PC
  - Ubuntu 20.04 LTS
  - RAM: 8GB
  - Intel(R) Core(TM) i7-6700 - 3.4GHz
  - 64 bits

- Security Level: FireSaber (AES-256)
Round 2

- KEY GENERATION: 1458.00 $\mu$ seconds
- ENCRYPTION: 1584.04 $\mu$ seconds
- DECRYPTION: 382.43 $\mu$ seconds

Round 3

- KEY GENERATION: 1970.18 $\mu$ seconds
- ENCRYPTION: 2435.74 $\mu$ seconds
- DECRYPTION: 574.68 $\mu$ seconds

* Round 2 had better performance
Saber - Average Time - ARM Architecture

● Round 2
  ○ KEY GENERATION:
    ■ 894.20 $\mu$ seconds
  ○ ENCRYPTION:
    ■ 753.70 $\mu$ seconds
  ○ DECRYPTION:
    ■ 211.09 $\mu$ seconds

● Round 3
  ○ KEY GENERATION:
    ■ 333.96 $\mu$ seconds
  ○ ENCRYPTION:
    ■ 355.25 $\mu$ seconds
  ○ DECRYPTION:
    ■ 128.25 $\mu$ seconds

* Round 3 had better performance
Saber - Bottlenecks - x64 Architecture

- **Round 2**
  - KEY GENERATION:
    - MatrixVectorMulti Function (81% Consumption)
  - ENCRYPTION:
    - MatrixVectorMulti Function (59% Consumption)
  - DECRYPTION:
    - InnerProd Function (95% Consumption)

- **Round 3**
  - KEY GENERATION:
    - MatrixVectorMulti Function (86% Consumption)
  - ENCRYPTION:
    - MatrixVectorMulti Function (68% Consumption)
  - DECRYPTION:
    - InnerProd Function (96% Consumption)

* MatrixVectorMulti and InnerProd are bottlenecks
Saber - Bottlenecks - ARM Architecture

● Round 2
  ○ KEY GENERATION:
    ■ MatrixVectorMulti Function (67% Consumption)
  ○ ENCRYPTION:
    ■ MatrixVectorMulti Function (40% Consumption)
  ○ DECRYPTION:
    ■ InnerProd Function (88% Consumption)

● Round 3
  ○ KEY GENERATION:
    ■ MatrixVectorMulti Function (64% Consumption)
  ○ ENCRYPTION:
    ■ MatrixVectorMulti Function (55% Consumption)
  ○ DECRYPTION:
    ■ InnerProd Function (89% Consumption)

* MatrixVectorMulti and InnerProd are bottlenecks
* Consumption values were more balanced
Saber Round 2 - x64 versus ARM Architectures
Saber Round 3 - x64 versus ARM Architectures

- x64 better 4 times and ARM better 26 times
Saber Round 3 Code Improvement

- Improvement in MatrixVectorMulti Function
  - Use shift operations instead of division by 2 on karatsuba_simple function that is inside MatrixVectorMulti function.
- What was better in performance?
  - Function had an improvement of 3.26% compared to Round 3 original code.
- Is improvement conclusive?
  - Tests showed better performance, however we can't affirm it's conclusive.
  - There are compilers that automatically change division by 2 to shift operations.
  - We suggest Saber team to evaluate this improvement and conclude if it really improved performance.
Kyber Test Devices

- **Mobile Device ARMv8**
  - Android 10
  - RAM: 8GB
  - Octa-core (2x2.73 GHz Mongoose M5 + 2x2.60 GHz Cortex-A76 + 4x2.0 GHz Cortex-A55)

- **PC**
  - Ubuntu 20.04 LTS
  - RAM: 8GB
  - Intel(R) Core(TM) i7-6700 - 3.4GHz
  - 64 bits

- Security Level: **Kyber1024 (AES-256)**
Kyber - NIST round 3

- Galaxy S20
- Linux
Average time spend in one loop

- Keypair: 9%
- Enc: 25%
- Dec: 66%
Kyber - Analysis Round 2 versus Round 3

● **Analysis**
  ○ Average execution time for Linux was increased
  ○ Average execution time for Android was reduced
  ○ Top values were kept

● **Conclusion**
  ○ Kyber was optimized for ARM architecture in newest NIST submission
Searching for code improvements

- Look for multiplication and division operations that could be replaced by bit shifting
  - Not found an effective change
- Use 90s variant to find out improvements
  - “The 90s variant of Kyber uses symmetric primitives that are standardized by NIST and accelerated in hardware on a large variety of platforms.” (Kyber, 2020)
  - For Galaxy S20 was not effective (see next slide), it increased the average execution time in 41.28%
    - Worst times for key pair generation and encryption
    - Better times for decryption
Kyber 1024 versions

- Kyber 1024
- Kyber 1024 90's