SABER: Module-LWR based KEM
Round 2

J. P. D’Anvers  A. Karmakar
S. S. Roy  F. Vercauteren
KU Leuven
August 22, 2019
0 Outline

1 Introduction

2 Round 2 changes

3 Implementations

4 Conclusion
1 Outline

1 Introduction

2 Round 2 changes

3 Implementations

4 Conclusion
1 General LWE based scheme

Alice

\[ A \leftarrow \mathcal{U}(\mathbb{Z}_q^{l \times l}) \]
\[ s, e \leftarrow \text{small}(\mathbb{Z}_q^{l \times 1}) \]
\[ b = A \cdot s + e \]
\[ v = b' \cdot s \]
\[ m' = \left\lfloor \frac{2}{q} (v' - v) \right\rfloor \]

Bob

\[ b, A \]
\[ s', e', e'' \leftarrow \text{small}(\mathbb{Z}_q^{1 \times l}) \]
\[ b'^T = A^T \cdot s' + e' \]
\[ v'^T = b'^T \cdot s' + e'' + \frac{q}{2} m \]
1 SABER

- Module:
  - Polynomial ring $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 2^{13}$
  - Rank of module 2, 3, 4 depending on security level
  - Flexibility: only one polynomial multiplication
1 SABER

Alice

\[
\begin{align*}
A & \leftarrow \mathcal{U}(\mathbb{R}_q^{l \times l}) \\
\mathbf{s}, \mathbf{e} & \leftarrow \text{small}(\mathbb{R}_q^{l \times 1}) \\
\mathbf{b} & = A \cdot \mathbf{s} + \mathbf{e} \\
\mathbf{v} & = \mathbf{b}' \cdot \mathbf{s} \\
m' & = \left\lfloor \frac{2}{q} (\mathbf{v}' - \mathbf{v}) \right\rfloor
\end{align*}
\]

Bob

\[
\begin{align*}
\mathbf{b}, A & \rightarrow \mathbf{s}', \mathbf{e}', \mathbf{e}'' \leftarrow \text{small}(\mathbb{R}_q^{1 \times l}) \\
\mathbf{b}'^T & = \mathbf{A}^T \cdot \mathbf{s}' + \mathbf{e}' \\
\mathbf{v}'^T & = \mathbf{b}'^T \cdot \mathbf{s}' + \mathbf{e}'' + \frac{q}{2} m
\end{align*}
\]
Module-LWR: SABER

- **Module:**
  - Polynomial ring $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 2^{13}$
  - Rank of module 2, 3, 4 depending on security level
  - Flexibility: only one polynomial multiplication

- **Learning with Rounding**
  - No generation of $e, e', e''$
  - Efficient bandwidth usage
1 SABER

Alice

\[ \mathbf{A} \leftarrow \mathcal{U}(R_{q}^{l \times l}) \]
\[ \mathbf{s} \leftarrow \text{small}(R_{q}^{l \times 1}) \]
\[ \mathbf{b} = \left\lfloor \frac{p}{q} \mathbf{A} \cdot \mathbf{s} \right\rfloor \]
\[ \mathbf{v} = \mathbf{b}' \cdot \mathbf{s} \]
\[ m' = \left\lfloor \frac{2}{q} (\mathbf{v}' - \frac{p}{T} \mathbf{v}) \right\rfloor \]

Bob

\[ \mathbf{b}', \mathbf{v}' \]
\[ \mathbf{b}', \mathbf{v}' \]
\[ \mathbf{b}' = \mathbf{s}' \leftarrow \text{small}(R_{q}^{1 \times l}) \]
\[ \mathbf{b}' = \left\lfloor \frac{p}{q} \mathbf{A}^{T} \cdot \mathbf{s}' \right\rfloor \]
\[ \mathbf{v}' = \left\lfloor \frac{T}{p} \mathbf{b}^{T} \cdot \mathbf{s}' + \frac{T}{2} m \right\rfloor \]
1 Module-LWR: SABER

▶ Module:
  • Polynomial ring $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 2^{13}$
  • Rank of module 2, 3, 4 depending on security level
  ⊕ Flexibility: only one polynomial multiplication

▶ Learning with Rounding
  ⊕ no generation of $e, e', e''$
  ⊕ efficient bandwidth usage

▶ power-of-two
  ⊕ easy sampling
  ⊕ no modular arithmetic
  ⊕ easy rounding = add constant and chop
  ⊕ no NTT for fast multiplication
  ⊕ Toom-Cook
  ⊕ easier masking
1  SABER

### Alice

\[
\begin{align*}
\textbf{A} & \leftarrow \mathcal{U}(R_q^{l \times l}) \\
\textbf{s} & \leftarrow \text{small}(R_q^{l \times 1}) \\
b & = (\textbf{A} \cdot \textbf{s} + \textbf{h}) \gg \log_2 \left( \frac{q}{p} \right) \\
v & = b' \cdot s \\
m' & = \left\lfloor \frac{2}{p}(v' - \frac{p}{T}v) \right\rfloor
\end{align*}
\]

### Bob

\[
\begin{align*}
\textbf{s}' & \leftarrow \text{small}(R_q^{1 \times l}) \\
b'^T & = (\textbf{A}^T \cdot \textbf{s}' + \textbf{h}) \gg \log_2 \left( \frac{q}{p} \right) \\
v'^T & = (b'^T \cdot \textbf{s}' + h_1 + \frac{p}{2}m) \gg \log_2 \left( \frac{p}{T} \right)
\end{align*}
\]
1 SABER

- binomial secret distribution
  - easy sampling
1 SABER

- binomial secret distribution
  - easy sampling
- No error correcting code
  - simpler implementation
  - easier masking
1 SABER - parameters

- $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 2^{13}$
- public key / ciphertext in $R_p$ and $R_T$ with $p = 2^{10}$ and $T = 2^4$
- Centered binomial distribution with 8 coins ($[-4, 4]$)
1  SABER - parameters

- \( R_q = \mathbb{Z}_q[X]/(X^{256} + 1) \) with \( q = 2^{13} \)
- Public key / ciphertext in \( R_p \) and \( R_T \) with \( p = 2^{10} \) and \( T = 2^4 \)
- Centered binomial distribution with 8 coins \([-4, 4]\)
- IND-CCA secure KEM version using FO-transformation

\[ R_q = \mathbb{Z}_q[X]/(X^{256} + 1) \]  with  \( q = 2^{13} \)
1 SABER - parameters

- $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 2^{13}$
- Public key / ciphertext in $R_p$ and $R_T$ with $p = 2^{10}$ and $T = 2^4$
- Centered binomial distribution with 8 coins ($[-4, 4]$)

- IND-CCA secure KEM version using FO-transformation

- Public Key: 992 Bytes
- Ciphertext: 1088 Bytes
- Failure probability: $2^{-136}$
- Security: 185 bits
# 1 SABER

<table>
<thead>
<tr>
<th>Sec Cat</th>
<th>fail prob</th>
<th>Classical</th>
<th>Quantum</th>
<th>pk (B)</th>
<th>sk (B)</th>
<th>ciphertext (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LightSaber-KEM:</strong> $k = 2$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^3$, $\mu = 10$</td>
<td>$2^{-120}$</td>
<td>126</td>
<td>115</td>
<td>672</td>
<td>1568</td>
<td>736</td>
</tr>
<tr>
<td><strong>Saber-KEM:</strong> $k = 3$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^4$, $\mu = 8$</td>
<td>$2^{-136}$</td>
<td>199</td>
<td>181</td>
<td>992</td>
<td>2304</td>
<td>1088</td>
</tr>
<tr>
<td><strong>FireSaber-KEM:</strong> $k = 4$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^6$, $\mu = 6$</td>
<td>$2^{-165}$</td>
<td>270</td>
<td>246</td>
<td>1312</td>
<td>3040</td>
<td>1472</td>
</tr>
</tbody>
</table>

Table: Security and correctness of Saber.KEM.
2 Outline

1 Introduction

2 Round 2 changes

3 Implementations

4 Conclusion
2 Changes for Round 2

- Generation of matrix $A$
2 Changes for Round 2

- Generation of matrix $A$
  - multiplication with $A$ and $A^T$
  - just-in-time possible for $A$
  - speed-up preferred in encryption
2 Serial vs parallel generation of A

- software
  - Keccak-Absorb() is more expensive than Keccak-Extract()
  - Hence, serial SHAKE is faster on non-vectorized microcontrollers
  - But, slower on Intel AVX
2 Serial vs parallel generation of $A$

- **software**
  - Keccak-Absorb() is more expensive than Keccak-Extract()
  - Hence, serial SHAKE is faster on non-vectorized microcontrollers
  - But, slower on Intel AVX

- **hardware**
  - Keccak core consumes 33% of overall area [BPC19] (including memory)
  - Keccak-Extract produces RND every 28 cycles
  - Polynomial multiplier consumes RND much slower than Keccak can produce
  - Serial Keccak makes implementation simpler
2 Changes for Round 2

- Generation of matrix $A$
2 Changes for Round 2

- Generation of matrix $A$
- Rounding $= \text{add constant} + \text{chopping}$
- one of the constants changed for security proof
2 Changes for Round 2

- Generation of matrix $A$

- Rounding $= \text{add constant} + \text{chopping}$
- one of the constants changed for security proof

- (Debated) smaller secret variance
  - e.g. trinary binomial distribution
  - would reduce public key and ciphertext size with $\pm 10\%$
  - too aggressive
3 Outline

1 Introduction

2 Round 2 changes

3 Implementations

4 Conclusion
3 Software Implementations

- Haswell AVX2 (KU Leuven, Belgium [DKRV18])
  - IND-CCA encapsulation/decapsulation 122K, 120K cycles
3 Software Implementations

- Haswell AVX2 (KU Leuven, Belgium [DKRV18])
  - IND-CCA encapsulation/decapsulation 122K, 120K cycles

- ARM Cortex-M (KU Leuven, Belgium [KMRV18])
  - Cortex-M4 (Speed)
    - encapsulation/decapsulation 1444 / 1543 K cycles
  - Cortex-M4 (Speed / Memory)
    - encapsulation/decapsulation 1530 / 1635 K cycles
    - encapsulation/decapsulation 7019 / 8115 bytes memory
  - Cortex-M0 (Memory)
    - encapsulation/decapsulation 6328 / 7509 K cycles
    - encapsulation/decapsulation 5119 / 6215 bytes memory
3 Hardware Implementations I

- High-speed HW (University of Birmingham, UK)
  - Instruction-set coprocessor architecture with all SABER components on HW
  - Generic HDL code: suitable for ASIC and FPGA implementation
  - IND-CPA encryption/decryption = 6/1.6 K cycles
  - IND-CCA encapsulation/decapsulation = \( \approx \frac{7}{8.5} \) K cycles
3 Hardware Implementations I

- High-speed HW (University of Birmingham, UK)
  - Instruction-set coprocessor architecture with all SABER components on HW
  - Generic HDL code: suitable for ASIC and FPGA implementation
  - IND-CPA encryption/decryption = 6/1.6 K cycles
  - IND-CCA encapsulation/decapsulation = ≈ 7/8.5 K cycles

- Lightweight HW/SW codesign (KU Leuven, Belgium)
  - Encapsulation/decapsulation require ≈ 4.2 ms
3 Hardware Implementations I

- High-speed HW (University of Birmingham, UK)
  - Instruction-set coprocessor architecture with all SABER components on HW
  - Generic HDL code: suitable for ASIC and FPGA implementation
  - IND-CPA encryption/decryption = 6/1.6 K cycles
  - IND-CCA encapsulation/decapsulation = ≈ 7/8.5 K cycles

- Lightweight HW/SW codesign (KU Leuven, Belgium)
  - Encapsulation/decapsulation require ≈ 4.2 ms

- High-speed HW/SW codesign (George Mason University, USA / Military University of Technology, Poland [HOKG18])
  - Encapsulation/decapsulation require ≈ 0.069 ms
3 Hardware Implementations II

- ASIC implementation (Tsinghua University, China)
  - Still in development
  - Polynomial multiplication
  - Area: 220626 $\mu m^2$ (307193GE)
  - Max Freq: 400 MHz
  - Power: 4.34 mW
3 Masking

- First order masking can be achieved by arithmetic masking in polynomial multiplication and Boolean masking for decoding.
- Saber uses power-of-two modulus
- Thus masking methods can be combined by Debraize’s arithmetic to boolean conversion [Deb12]
- Time with masking roughly doubles.
4 Outline

1 Introduction

2 Round 2 changes

3 Implementations

4 Conclusion
4 Conclusion

SABER is:

- Flexible
4 Conclusion

SABER is:

▶ Flexible
▶ Simple

▶ More work in the pipeline
4 Conclusion

SABER is:

- Flexible
- Simple
- Efficient
4 Conclusion

SABER is:

- Flexible
- Simple
- Efficient

- More work in the pipeline
4 References I

Utsav Banerjee, Abhishek Pathak, and Anantha P. Chandrakasan.
An Energy-Efficient Configurable Lattice Cryptography Processor for the Quantum-Secure Internet of Things.

Blandine Debraize.
Efficient and provably secure methods for switching from arithmetic to boolean masking.

Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM.
4 References II


Questions?