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 Outline

1 General LWE based scheme

$$
\begin{array}{lcl}
\text{Alice} & & \text{Bob} \\
\\
\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times l}) & & \\
\boldsymbol{s}, \boldsymbol{e} \leftarrow \mathsf{small}(\mathbb{Z}_q^{l\times 1}) & & \\
\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e} & \xrightarrow{\boldsymbol{b}, \boldsymbol{A}} & \boldsymbol{s'}, \boldsymbol{e'}, \boldsymbol{e''} \leftarrow \mathsf{small}(\mathbb{Z}_q^{1\times l}) \\
 & & \boldsymbol{b}'^T = \boldsymbol{A}^T \cdot \boldsymbol{s'} + \boldsymbol{e'} \\
v = \boldsymbol{b}' \cdot \boldsymbol{s} & \xleftarrow{\boldsymbol{b}', v'} & v'^T = \boldsymbol{b}^T \cdot \boldsymbol{s'} + \boldsymbol{e''} + \frac{q}{2} m \\
m' = \lfloor \frac{2}{q}(v' - v) \rceil & &
\end{array}
$$

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		Bob
$\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$		
$\boldsymbol{s}, \boldsymbol{e} \leftarrow \mathsf{small}(R_q^{l\times 1})$		
$\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e}$	$\xrightarrow{\boldsymbol{b}, \boldsymbol{A}}$	$\boldsymbol{s}', \boldsymbol{e}', \boldsymbol{e}'' \leftarrow \mathsf{small}(R_q^{1\times l})$
		$\boldsymbol{b}'^T = \boldsymbol{A}^T \cdot \boldsymbol{s}' + \boldsymbol{e}'$
$v = \boldsymbol{b}' \cdot \boldsymbol{s}$	$\xleftarrow{\boldsymbol{b}', v'}$	$v'^T = \boldsymbol{b}^T \cdot \boldsymbol{s}' + \boldsymbol{e}'' + \frac{q}{2} m$
$m' = \lfloor \frac{2}{q}(v' - v) \rceil$		

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 $\boldsymbol{e}, \boldsymbol{e}', \boldsymbol{e}''$
  - ⊕ Efficient bandwidth usage

1 SABER

Alice		Bob
$\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$		
$\boldsymbol{s} \leftarrow \text{small}(R_q^{l\times 1})$		
$\boldsymbol{b} = \lfloor \frac{p}{q}\boldsymbol{A}\cdot\boldsymbol{s} \rceil$	$\xrightarrow{\boldsymbol{b},\,\boldsymbol{A}}$	$\boldsymbol{s'} \leftarrow \text{small}(R_q^{1\times l})$
		$\boldsymbol{b}'^T = \lfloor \frac{p}{q}\boldsymbol{A}^T\cdot\boldsymbol{s'} \rceil$
$v = \boldsymbol{b}'\cdot\boldsymbol{s}$	$\xleftarrow{\boldsymbol{b}',\,v'}$	$v'^T = \lfloor \frac{T}{p}\boldsymbol{b}^T\cdot\boldsymbol{s'} + \frac{T}{2}m \rceil$
$m' = \lfloor \frac{2}{q}(v' - \frac{p}{T}v) \rceil$		

KU LEUVEN

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 $\boldsymbol{e}, \boldsymbol{e}', \boldsymbol{e}''$
  - ⊕ 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		Bob
$\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{l\times l})$		
$\boldsymbol{s} \leftarrow \mathsf{small}(R_q^{l\times 1})$		
$\boldsymbol{b} = (\boldsymbol{A}\cdot\boldsymbol{s} + \boldsymbol{h}) \gg \log_2(\frac{q}{p})$	$\xrightarrow{\boldsymbol{b},\boldsymbol{A}}$	$\boldsymbol{s}' \leftarrow \mathsf{small}(R_q^{1\times l})$
		$\boldsymbol{b}'^T = (\boldsymbol{A}^T\cdot\boldsymbol{s}' + \boldsymbol{h}) \gg \log_2(\frac{q}{p})$
$v = \boldsymbol{b}'\cdot\boldsymbol{s}$	$\xleftarrow{\boldsymbol{b}', v'}$	$v'^T = (\boldsymbol{b}^T\cdot\boldsymbol{s}' + h_1 + \frac{p}{2}m) \gg \log_2(\frac{p}{T})$
$m' = \lfloor \frac{2}{p}(v' - \frac{p}{T}v) \rceil$		

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

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

Sec Cat	fail prob	Classical	Quantum	pk (B)	sk (B)	ciphertext (B)
LightSaber-KEM: $k = 2$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^3$, $\mu = 10$						
1	2^{-120}	126	115	672	1568	736
Saber-KEM: $k = 3$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^4$, $\mu = 8$						
3	2^{-136}	199	181	992	2304	1088
FireSaber-KEM: $k = 4$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^6$, $\mu = 6$						
5	2^{-165}	270	246	1312	3040	1472

Table: Security and correctness of Saber.KEM.

2 Outline

2 Changes for Round 2

- Generation of matrix $\boldsymbol{A}$

2 Changes for Round 2

- Generation of matrix $\boldsymbol{A}$
  - multiplication with $\boldsymbol{A}$ and $\boldsymbol{A}^T$
  - just-in-time possible for $\boldsymbol{A}$
  - 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 $\boldsymbol{A}$

2 Changes for Round 2

- Generation of matrix $\boldsymbol{A}$

- Rounding = add constant + chopping
- one of the constants changed for security proof

2 Changes for Round 2

- Generation of matrix $\boldsymbol{A}$

- Rounding $=$ add constant $+$ 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

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 = $\approx 7/8.5$ 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 = $\approx 7/8.5$ 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 = $\approx 7/8.5$ 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 um^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

4 Conclusion

SABER is:

- Flexible

4 Conclusion

SABER is:

- Flexible
- Simple

4 Conclusion

SABER is:

- Flexible
- Simple
- Efficient

4 Conclusion

SABER is:

- Flexible
- Simple
- Efficient

- More work in the pipeline

KU LEUVEN

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.
In *IEEE International Solid-State Circuits Conference*, pages 46–48, 2019.

Blandine Debraize.
Efficient and provably secure methods for switching from arithmetic to boolean masking.
In *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 LNCS, 2012.

Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren.
Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM.
In *AFRICACRYPT 2018*, pages 282–305, 2018.

4 References II

James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu.
Standard Lattice-Based Key Encapsulation on Embedded Devices.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018, 8 2018.

Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede.
Saber on ARM: CCA-secure module lattice-based key encapsulation on ARM.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018, 8 2018.

KU LEUVEN

4 Questions?