KU LEUVEN

SABER: Module-LWR based KEM

J. P. D'Anvers A. Karmakar S. S. Roy F. Vercauteren

imec - COSIC

- Setup: modulus $q \in \mathbb{Z}$, dimension l

- Setup: modulus $q \in \mathbb{Z}$, dimension l
- LWE: samples of the form

$$\left(\boldsymbol{a}, b = \boldsymbol{a}^T \boldsymbol{s} + e\right) \in \mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_q$$

  - $\boldsymbol{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1})$, uniform random for each sample
  - secret vector $\boldsymbol{s} \in \mathbb{Z}_q^{l\times 1}$ fixed for all samples
  - $e \leftarrow \chi(\mathbb{Z}_q)$ small error

- Setup: modulus $q \in \mathbb{Z}$, dimension l
- LWE: samples of the form

$$\left(\boldsymbol{a}, b = \boldsymbol{a}^T \boldsymbol{s} + e\right) \in \mathbb{Z}_q^{l \times 1} \times \mathbb{Z}_q$$

  - $\boldsymbol{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l \times 1})$, uniform random for each sample
  - secret vector $\boldsymbol{s} \in \mathbb{Z}_q^{l \times 1}$ fixed for all samples
  - $e \leftarrow \chi(\mathbb{Z}_q)$ small error
- LWR: LWE with deterministic noise due to scaling and rounding

$$\left(\boldsymbol{a}, b = \left\lfloor \frac{p}{q}(\boldsymbol{a}^T \boldsymbol{s}) \right\rceil\right) \in \mathbb{Z}_q^{l \times 1} \times \mathbb{Z}_p$$

  - q/p determines inherent noise

- Replace $\mathbb{Z}_q$ by larger ring $R_q = \mathbb{Z}_q[X]/(f(X))$
- Dimension l is now product of module rank k and $\deg(f)$

LWE	Module-LWE
$l = 768$	$k = 3$ and $f(X) = x^{256} + 1$
$\boldsymbol{a}^T = [a_0, a_1, \ldots, a_{767}]$	$\boldsymbol{a}^T = [\tilde{a}_0, \tilde{a}_1, \tilde{a}_2], \tilde{a}_i \in R_q$
$\boldsymbol{s}^T = [s_0, s_1, \ldots, s_{767}]$	$\boldsymbol{s}^T = [\tilde{s}_0, \tilde{s}_1, \tilde{s}_2], \tilde{s}_i \in R_q$
$b = \boldsymbol{a}^T \boldsymbol{s} + e \in \mathbb{Z}_q$	$\tilde{b} = \boldsymbol{a}^T \boldsymbol{s} + \tilde{e} \in R_q$

- Module-LWR: scaling and rounding coefficient-wise

SABER Parameter Choices

- Simplicity: moduli $p|q$ are powers of 2
  - $\oplus$ all security levels $p = 2^{10}$ and $q = 2^{13}$
  - $\oplus$ easy sampling
  - $\oplus$ no modular arithmetic
  - $\oplus$ easy rounding = add constant and chop
  - $\oplus$ scaling is uniform
  - $\ominus$ no NTT for fast multiplication
  - $\oplus$ Toom-Cook: division by 8, so work mod 2^{16} to multiply

SABER Parameter Choices

- Simplicity: moduli $p|q$ are powers of 2
  - $\oplus$ all security levels $p = 2^{10}$ and $q = 2^{13}$
  - $\oplus$ easy sampling
  - $\oplus$ no modular arithmetic
  - $\oplus$ easy rounding = add constant and chop
  - $\oplus$ scaling is uniform
  - $\ominus$ no NTT for fast multiplication
  - $\oplus$ Toom-Cook: division by 8, so work mod 2^{16} to multiply
- Only one polynomial ring $R_q = \mathbb{Z}_q[x]/(x^{256} + 1)$ with $q = 2^{13}$

- Simplicity: moduli $p|q$ are powers of 2
  - $\oplus$ all security levels $p = 2^{10}$ and $q = 2^{13}$
  - $\oplus$ easy sampling
  - $\oplus$ no modular arithmetic
  - $\oplus$ easy rounding = add constant and chop
  - $\oplus$ scaling is uniform
  - $\ominus$ no NTT for fast multiplication
  - $\oplus$ Toom-Cook: division by 8, so work mod 2^{16} to multiply
- Only one 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

- Simplicity: moduli $p|q$ are powers of 2
  - $\oplus$ all security levels $p = 2^{10}$ and $q = 2^{13}$
  - $\oplus$ easy sampling
  - $\oplus$ no modular arithmetic
  - $\oplus$ easy rounding = add constant and chop
  - $\oplus$ scaling is uniform
  - $\ominus$ no NTT for fast multiplication
  - $\oplus$ Toom-Cook: division by 8, so work mod 2^{16} to multiply
- Only one 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
- Secrets sampled from centered binomial distribution β_μ
  - Sample easy as $\sum_{i=1}^{\mu} a_i - \sum_{i=1}^{\mu} b_i$ with $a_i, b_i \in \{0, 1\}$

Alice

A = gen(seed_A)

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \left\lfloor \frac{p}{q} \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \right\rceil$$

Bob

A^T = gen(seed_A)^T

$$\begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \left\lfloor \frac{p}{q} \begin{bmatrix} a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \\ a_3 & a_6 & a_9 \end{bmatrix} \begin{bmatrix} s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} \right\rceil$$

$$\xrightarrow{\text{seed\_A},\, b_1, b_2, b_3}$$

$$\xleftarrow{b'_1, b'_2, b'_3}$$

$$k = \begin{bmatrix} b'_1 & b'_2 & b'_3 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

$$k' = \begin{bmatrix} s'_1 & s'_2 & s'_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$k \sim k'$$

Fix up difference using reconciliation info

- Equivalent of standard Regev-type LWE encryption

- Equivalent of standard Regev-type LWE encryption
- KeyGen:
  - $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$, $\boldsymbol{A} \leftarrow \texttt{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{k \times k}$
  - $\boldsymbol{s} \leftarrow \beta_\mu(R_q^{k \times 1})$, $\boldsymbol{b} = \texttt{chop}(\boldsymbol{As} + \boldsymbol{h}, q, p) \in R_p^{k \times 1}$
  - $pk := (\boldsymbol{b}, seed_{\boldsymbol{A}}), sk := \boldsymbol{s}$

- Equivalent of standard Regev-type LWE encryption
- KeyGen:
  - $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$, $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{k \times k}$
  - $\boldsymbol{s} \leftarrow \beta_\mu(R_q^{k \times 1})$, $\boldsymbol{b} = \mathtt{chop}(\boldsymbol{As} + \boldsymbol{h}, q, p) \in R_p^{k \times 1}$
  - $pk := (\boldsymbol{b}, seed_{\boldsymbol{A}}), sk := \boldsymbol{s}$
- Encrypt: $pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), m \in \mathcal{M}; r$
  - $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{k \times l}$
  - $\boldsymbol{s'} \leftarrow \beta_\mu(R_q^{k \times 1})$, $\boldsymbol{b'} = \mathtt{chop}(\boldsymbol{A}^T \boldsymbol{s'} + \boldsymbol{h}, q, p) \in R_p^{k \times 1}$
  - $v' = \boldsymbol{b}^T \boldsymbol{s'} \in R_p$
  - $c_m = \mathtt{chop}(v' + h_1 + \frac{p}{2} m, p, 2t) \in R_{2t}$
  - $c := (c_m, \boldsymbol{b'})$

- Equivalent of standard Regev-type LWE encryption
- KeyGen:
  - $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$, $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{k\times k}$
  - $\boldsymbol{s} \leftarrow \beta_\mu(R_q^{k\times 1})$, $\boldsymbol{b} = \mathsf{chop}(\boldsymbol{A}\boldsymbol{s} + \boldsymbol{h}, q, p) \in R_p^{k\times 1}$
  - $pk := (\boldsymbol{b}, seed_{\boldsymbol{A}}), sk := \boldsymbol{s}$
- Encrypt: $pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), m \in \mathcal{M}; r$
  - $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{k\times l}$
  - $\boldsymbol{s'} \leftarrow \beta_\mu(R_q^{k\times 1})$, $\boldsymbol{b'} = \mathsf{chop}(\boldsymbol{A}^T\boldsymbol{s'} + \boldsymbol{h}, q, p) \in R_p^{k\times 1}$
  - $v' = \boldsymbol{b}^T\boldsymbol{s'} \in R_p$
  - $c_m = \mathsf{chop}(v' + h_1 + \frac{p}{2}m, p, 2t) \in R_{2t}$
  - $c := (c_m, \boldsymbol{b'})$
- Decrypt: $sk = \boldsymbol{s}, (c_m, \boldsymbol{b'})$
  - $v = \boldsymbol{b'}^T\boldsymbol{s} \in R_p$
  - $m' = \mathsf{chop}(v - \frac{p}{2t}c_m + h_2, p, 2) \in R_2$

- KeyGen: same as before, but sk includes $z \in \mathcal{U}(\{0,1\}^{256})$
- Encaps: $pk = (\boldsymbol{b}, seed_{\boldsymbol{A}})$
  - $m \leftarrow \mathcal{U}(\{0,1\}^{256})$
  - $(\hat{K}, r) = \mathcal{G}(pk, m)$
  - $c = \texttt{Saber.Enc}(pk, m; r)$
  - $K = \mathcal{H}(\hat{K}, c)$
  - return (c, K)
- Decaps: $sk = (\boldsymbol{s}, z), pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), c$
  - $m' = \texttt{Saber.Dec}(\boldsymbol{s}, c)$
  - $(\hat{K}', r') = \mathcal{G}(pk, m')$
  - $c' = \texttt{Saber.Enc}(pk, m'; r')$
  - if $c = c'$ return $K = \mathcal{H}(\hat{K}', c)$
  - else return $K = \mathcal{H}(z, c)$

Common parameters: $q = 2^{13}$, $p = 2^{10}$, $f(x) = x^{256} + 1$

cat	failure	attack	class.	quant.	pk (B)	sk* (B)	ct (B)
LightSaber-KEM: $k = 2$, $t = 2^2$, $\mu = 5$							
1	2^{-120}	primal dual	126 126	115 115	672	992	736
Saber-KEM: $k = 3$, $t = 2^3$, $\mu = 4$							
3	2^{-136}	primal dual	199 198	181 180	992	1344	1088
FireSaber-KEM: $k = 4$, $t = 2^5$, $\mu = 3$							
5	2^{-165}	primal dual	270 270	246 245	1312	1760	1472

* includes the public key

Platform: Intel(R) Core(TM) i7-6600U, 2.60GHz with hyper-threading, Turbo-Boost, and multi-core support disabled

Saber-KEM: `C implementation`

- Keygen: 190,420 cycles
- Encaps: 279,291 cycles
- Decaps: 306,346 cycles

Saber-KEM: `AVX2 optimized` (can be improved)

- Keygen: 101,138 cycles
- Encaps: 125,392 cycles
- Decaps: 129,138 cycles

- $\oplus$ Easy to implement: no modular reductions, no rounding, no rejection sampling, no random lifting
- $\oplus$ Modular structure and flexibility: only need to implement arithmetic in R_q with $q = 2^{13}$ and $f(x) = x^{256} + 1$
- $\oplus$ Less randomness since no error sampling as for LWE
- $\oplus$ Low bandwith: more compact than LWE-based schemes, even with compression

SABER KEM $\oplus$ and $\ominus$

- $\oplus$ Easy to implement: no modular reductions, no rounding, no rejection sampling, no random lifting
- $\oplus$ Modular structure and flexibility: only need to implement arithmetic in R_q with $q = 2^{13}$ and $f(x) = x^{256} + 1$
- $\oplus$ Less randomness since no error sampling as for LWE
- $\oplus$ Low bandwith: more compact than LWE-based schemes, even with compression
- $\ominus$ No NTT, so resort to Karatsuba and Toom-Cook
- $\ominus$ No signature scheme

S
A
B
E
R

Simple
A
B
E
R

Secure

A

B

E

R

Secure
Adaptable
B
E
R

Secure
Adaptable
Belgian
E
R

Secure
Adaptable
Belgian
Efficient
R

Secure
Adaptable
Belgian
Efficient
Rounding Trumps* all . . .

* Research not funded by the Donald