Adding Distributed Decryption and Key Generation to a Ring-LWE Based CCA Encryption Scheme

N.P. Smart

COSIC,
KU Leuven, ESAT,
Kasteelpark Arenberg 10, bus 2452,
B-3001 Leuven-Heverlee,
Belgium.

Joint work with Michael Kraitsberg, Yehuda Lindell, Valery Osheter, and Younes Talibi Alaoui

February 18, 2019

Distributed Decryption

Adding threshold capability to any IND-CCA encryption scheme is problematic

- Cannot release the plaintext in clear until the CCA check is complete

For post-quantum schemes this becomes more complex

- PQC schemes not particularly well suited to distributed decryption

Distributed Decryption

We propose to do this for a LWE based PQC scheme.

- Using a combination of various MPC technologies
- GC and LSSS
- ISN and Shamir secret sharing

Tailor the MPC to the specific situation

The LIMA Scheme: KeyGen

1. $a = (a_0, \ldots, a_{N-1}) \underset{\text{XOF}}{\leftarrow} \mathbb{F}_q^N$.
2. For $i = 0$ to $N - 1$ do $s_i \leftarrow \text{GenerateGaussianNoise}_{\text{XOF}}(\sigma)$.
3. For $i = 0$ to $N - 1$ do $e'_i \leftarrow \text{GenerateGaussianNoise}_{\text{XOF}}(\sigma)$.
4. $\mathbf{a} \leftarrow \text{FFT}(a)$, $\mathbf{s} \leftarrow \text{FFT}(s)$, $\mathbf{e}' \leftarrow \text{FFT}(e')$.
5. $\mathbf{b} \leftarrow (\mathbf{a} \otimes \mathbf{s}) \oplus \mathbf{e}'$,
6. $\mathfrak{sk} \leftarrow (\mathbf{s}, \mathbf{a}, \mathbf{b})$.
7. $\mathfrak{pk} \leftarrow (\mathbf{a}, \mathbf{b})$.
8. Return $(\mathfrak{pk}, \mathfrak{sk})$

KeyGen

The random values produced in lines 2 and 3 use the following operation:

GenerateGaussianNoise$_{\mathsf{XOF}}(\sigma)$

1. $t \leftarrow \mathsf{XOF}[5]$; interpretting t as a bit string of length 40.
2. $s \leftarrow 0$.
3. For $i = 0$ to 19 do
   3.1 $s \leftarrow s - t[2 \cdot i] + t[2 \cdot i + 1]$.
4. Return s.

If we replace the XOF by producing a source of random bits, this means the KeyGen operation is totally linear

- As FFT is linear

This means creating a distributed KeyGen will be easy (see later).

The LIMA Scheme: Enc-CPA-Sub($\mathbf{m}, \mathfrak{pk}$, XOF)

1. $\ell = |\mathbf{m}|$.
2. If $\ell > N$ then return $\perp$.
3. $\mu \leftarrow$ BV-2-RE($\mathbf{m}$),
4. For $i = 0$ to $N - 1$ do $v_i \leftarrow \mathsf{GenerateGaussianNoise}_{\mathsf{XOF}}(\sigma)$.
5. For $i = 0$ to $N - 1$ do $e_i \leftarrow \mathsf{GenerateGaussianNoise}_{\mathsf{XOF}}(\sigma)$.
6. For $i = 0$ to $N - 1$ do $d_i \leftarrow \mathsf{GenerateGaussianNoise}_{\mathsf{XOF}}(\sigma)$.
7. $\mathbf{v} \leftarrow \mathsf{FFT}(v)$, $\mathbf{e} \leftarrow \mathsf{FFT}(e)$.
8. $x \leftarrow d + \Delta_q \cdot \mu \pmod{q}$.
9. $s \leftarrow \mathsf{FFT}^{-1}(\mathbf{b} \otimes \mathbf{v})$.
10. $t \leftarrow s + x$.
11. $c_0 \leftarrow \mathsf{Trunc}(t, \ell)$.
12. $\mathbf{c}_1 \leftarrow (\mathbf{a} \otimes \mathbf{v}) \oplus \mathbf{e}$.
13. Output $\mathbf{c} = (c_0, \mathbf{c}_1)$.

The LIMA Scheme: Dec-CPA($\mathbf{c}, \mathfrak{sk}$)

1. Define ℓ to be the length of c_0.
2. If $\ell \neq 0 \pmod 8$ then return $\perp$.
3. $v \leftarrow \mathrm{FFT}^{-1}(\mathbf{s} \otimes \mathbf{c}_1)$.
4. $t \leftarrow \mathrm{Trunc}(v, \ell)$.
5. $f \leftarrow c_0 - t$.
6. Convert f into centered-representation modulo q.
7. $\mu \leftarrow \left| \left\lfloor \frac{2}{q} f \right\rceil \right|$
8. $\mathbf{m} \leftarrow$ RE-2-BV(μ).
9. Return $\mathbf{m}$.

The LIMA Scheme: CCA version

The problem comes in the CCA version of the scheme:

Enc-CCA($\mathbf{m}, \mathfrak{pk}, \mathbf{r}$):

1. If $|\mathbf{r}| \neq 256$ or $|\mathbf{m}| \geq N - 256$ then return $\perp$.
2. $\mu \leftarrow \mathbf{m} \| \mathbf{r}$.
3. XOF $\leftarrow$ KMAC($\mu, 0x03, 0$).
4. $\mathbf{c} \leftarrow$ Enc-CPA-Sub($\mu, \mathfrak{pk}$, XOF).
5. Return $\mathbf{c}$.

The LIMA Scheme: CCA version

The problem comes in the CCA version of the scheme:

Dec-CCA($\mathbf{c}, \mathfrak{sk}$):

1. $\mu \leftarrow$ Dec-CPA($\mathbf{c}, \mathfrak{sk}$).
2. If $|\mu| < 256$ then return $\perp$.
3. XOF $\leftarrow$ KMAC($\mu, 0x03, 0$).
4. $\mathbf{c}' \leftarrow$ Enc-CPA-Sub($\mu, \mathfrak{pk}$, XOF).
5. If $\mathbf{c} \neq \mathbf{c}'$ then return $\perp$.
6. $\mathbf{m} \| \mathbf{r} \leftarrow \mu$, where $\mathbf{r}$ is 256 bits long.
7. Return $\mathbf{m}$.

We need to evaluate the KMAC (SHA-3) algorithm on μ before we release the $\mathbf{m}$ component of μ.

Distributed Decryption

We choose a three party, one active adversary, scenario

We share the secret key using Ito–Nishizeki–Saito sharing

In particular S_1 is assumed to hold $(\mathbf{s}_1^{1,2}, \mathbf{s}_1^{1,3}) \in \mathbb{Z}_q^N$, S_2 is assumed to hold $(\mathbf{s}_2^{1,2}, \mathbf{s}_1^{2,3}) \in \mathbb{Z}_q^N$, and S_2 is assumed to hold $(\mathbf{s}_2^{1,3}, \mathbf{s}_2^{2,3}) \in \mathbb{Z}_q^N$ such that

$$\mathbf{s}_1^{1,2} + \mathbf{s}_2^{1,2} = \mathbf{s}_1^{1,3} + \mathbf{s}_2^{1,3} = \mathbf{s}_1^{2,3} + \mathbf{s}_2^{2,3} = \mathbf{s}.$$

Round Function

We require a protocol which takes an ISN-sharing of a vector **f** and produces the output of the function

$$\mu \leftarrow \left| \left\lfloor \frac{2}{q} f \right\rceil \right|$$

This is done using a special actively secure GC protocol for the $(1, 3)$-threshold setting (see paper).

Requires one garbled circuit to be produced, of $262, 144$ AND gates.

This effectively gives us Dec-CPA.

SHA-3 Evaluation

Given the output of Dec-CPA we need to pass it into the XOF to get the output needed for the Enc-CPA-Sub routine.

This requires evaluating the SHA-3 round function a number of times.

- 38,400 AND gates per round

The rest of Enc-CPA-Sub becomes essentially locally computations as FFT is linear

Only need to produce the truncation of $d + \Delta_q \cdot \mu + x$ in a secure fashion for testing equality

- Also done with a garbled circuit

Distributed Decryption: Run Time

Despite one execute of a garbled SHA-3 round function taking only 16ms, the overall decryption time takes over 4 seconds!

Why?

The real problem is the round function having to be computed on *each coefficient*

In LWE schemes there are a lot of coefficients, the ring dimension.

Distributed KeyGen

Distributed Key Generation is much easier.

Here we use SCALE-MAMBA in Shamir $(1, 3)$ mode.

- An offline/online based MPC system
- Offline produces shared random Beaver triples (first two components are random)
- Offline phases allows production of shared random bits! (v. important for us)

Distributed KeyGen

As we can produce shared random bits, production of approximate discrete Gaussians is trivial...

<u>SecGauss()</u>

1. $[a] \leftarrow 0$.
2. For $i \in [0, \ldots, 19]$ do
   2.1 $[b] \leftarrow \text{Bits}$, $[b'] \leftarrow \text{Bits}$.
   2.2 $[a] \leftarrow [a] + [b] - [b']$.
3. Return $[a]$.

In fact this is (after the offline phase) a completely local computation.

Distributed KeyGen

From this distrbuted KeyGen is simply linear operations (FFTs) and then converting data to the ISN shared format.

The per-coefficient operation is given by

KG-Coeff(i)

1. $[\underline{s}]_i \leftarrow$ SecGauss(), $[\underline{e}]_i \leftarrow$ SecGauss().
2. $([\underline{s}_1^{1,2}]_i, [\underline{s}_1^{1,3}]_i, [c]) \leftarrow$ Triples, $([\underline{s}_1^{2,3}]_i, [b], [c]) \leftarrow$ Triples.
3. $[\underline{s}_2^{1,2}]_i \leftarrow [\underline{s}]_i - [\underline{s}_1^{1,2}]_i$, $[\underline{s}_2^{1,3}]_i \leftarrow [\underline{s}]_i - [\underline{s}_1^{1,3}]_i$, $[\underline{s}_2^{2,3}]_i \leftarrow [\underline{s}]_i - [\underline{s}_1^{2,3}]_i$.
4. Output-To$(1, [\underline{s}_1^{1,2}]_i)$, Output-To$(1, [\underline{s}_1^{1,3}]_i)$.
5. Output-To$(2, [\underline{s}_1^{2,3}]_i)$, Output-To$(2, [\underline{s}_2^{1,2}]_i)$.
6. Output-To$(3, [\underline{s}_2^{1,3}]_i)$, Output-To$(3, [\underline{s}_2^{2,3}]_i)$.

Distributed KeyGen

KeyGen()

1. All players agree on a key for a XOF XOF.
2. $\underline{a} \underset{\text{XOF}}{\leftarrow} \mathbb{F}_q^N$.
3. For $i \in [0, \ldots, N-1]$ execute KG-Coeff(i).
4. $[\underline{b}] \leftarrow \underline{a} \cdot [\underline{s}] + [\underline{e}] \pmod{\Phi_{2 \cdot N}(X)}$.
   This is a completely local operation as $\underline{a}$ is public
5. For $i \in [0, \ldots, N-1]$ execute Output($[\underline{b}]_i$).
6. $\mathbf{a} \leftarrow \mathrm{FFT}(\underline{a})$, $\mathbf{b} \leftarrow \mathrm{FFT}(\underline{b})$ [Again local operations]
7. $\mathfrak{pk} \leftarrow (\mathbf{a}, \mathbf{b})$.
8. Player S_1 executes $\mathbf{s}_1^{1,2} \leftarrow \mathrm{FFT}(\underline{s}_1^{1,2})$ and $\mathbf{s}_1^{1,3} \leftarrow \mathrm{FFT}(\underline{s}_1^{1,3})$.
9. Player S_2 executes $\mathbf{s}_2^{1,2} \leftarrow \mathrm{FFT}(\underline{s}_2^{1,2})$ and $\mathbf{s}_1^{2,3} \leftarrow \mathrm{FFT}(\underline{s}_1^{2,3})$.
10. Player S_3 executes $\mathbf{s}_2^{1,3} \leftarrow \mathrm{FFT}(\underline{s}_2^{1,3})$ and $\mathbf{s}_2^{2,3} \leftarrow \mathrm{FFT}(\underline{s}_2^{2,3})$.

KeyGen Runtime

We timed this with SCALE-MAMBA v1.2 and obtained a run time of 1.22 seconds

Of this one second was actually producing the output

- Due to SCALE-MAMBA doing IO in serial as opposed to parallel protocol.
- Requiring 6144 rounds as opposed to one.

Conclusions

We have shown that MPC can be used to produce distributed/threshold implementations of a PQC encryption scheme.

Runtimes are a little disappointing.

- Main issue is the large ring degree (1024) used in LIMA.

The problem is not in the CCA transform (i.e. the SHA-3 evaluation)

The use of FFT like operations is also not a problem as these are linear

- Assuming you split up the MPC operation in a sensible manner to exploit this.

Suggest looking at distributed/threshold capabilities as a potential secondary criteria in the NIST competition.

Any Questions?