

Implementing RLWE-based Schemes Using an RSA Co-Processor

Martin R. Albrecht¹, Christian Hanser², Andrea Hoeller²,
Thomas Pöppelmann³, **Fernando Virdia**¹, Andreas Wallner²

¹Information Security Group, Royal Holloway, University of London, UK

²Infineon Technologies Austria AG

³Infineon Technologies AG, Germany

21 October 2019
COSIC Seminar, KU Leuven

- 📄 [Sho97] introduces a fast¹ order-finding quantum algorithm that allows factoring and computing discrete logs in Abelian groups.
- 📄 Since then, there has been a growing effort to develop efficient public-key encryption and signature algorithms that can resist cryptanalysis using large-scale general quantum computers.

¹Let's not go there.

- 🔲 [Sho97] introduces a fast¹ order-finding quantum algorithm that allows factoring and computing discrete logs in Abelian groups.
- 🔲 Since then, there has been a growing effort to develop efficient public-key encryption and signature algorithms that can resist cryptanalysis using large-scale general quantum computers.
- 🔲 In 2016, the US National Institute of Standards and Technology (NIST) started a several year long process to standardise post-quantum cryptographic schemes [Nat16].
- 🔲 Many of the proposed schemes are based on problems defined over polynomial rings, such as the RLWE problem.

¹Let's not go there.

- 🧩 In practice, cryptographic schemes have two crucial requirements²: high performance and ease of deployment.
- 🧩 Optimised implementations are an active area of research.

²Other than being secure in some appropriate model!

- 🧩 In practice, cryptographic schemes have two crucial requirements²: high performance and ease of deployment.
- 🧩 Optimised implementations are an active area of research.
- 🧩 As part of the NIST process, designers often provided fast software implementations with a focus on modern CPU architectures.
- 🧩 Furthermore, a lot of work has been done in the direction of constrained (often embedded) environments, such as microcontrollers or *smart cards*.

²Other than being secure in some appropriate model!



Currently available smart-cards provide low-power 16-bit and 32-bit CPUs and small amounts of RAM.

- 📱 *Currently available* smart-cards provide low-power 16-bit and 32-bit CPUs and small amounts of RAM.
- 📱 These are augmented with specific co-processors enabling them to run Diffie-Hellman key exchange (over finite fields and elliptic curves) and RSA encryption and signatures.

- 📦 *Currently available* smart-cards provide low-power 16-bit and 32-bit CPUs and small amounts of RAM.
- 📦 These are augmented with specific co-processors enabling them to run Diffie-Hellman key exchange (over finite fields and elliptic curves) and RSA encryption and signatures.
- 📦 For example, the SLE 78CLUF5000 Infineon chip card provides:
 - 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
 - AES and SHA256 co-processors³,
 - \mathbb{Z}_N adder and multiplier for $\log_2 N = 2200$ (“the RSA co-processor”).

³And DES!

- 📠 *Currently available* smart-cards provide low-power 16-bit and 32-bit CPUs and small amounts of RAM.
- 📠 These are augmented with specific co-processors enabling them to run Diffie-Hellman key exchange (over finite fields and elliptic curves) and RSA encryption and signatures.
- 📠 For example, the SLE 78CLUF5000 Infineon chip card provides:
 - 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
 - AES and SHA256 co-processors³,
 - \mathbb{Z}_N adder and multiplier for $\log_2 N = 2200$ (“the RSA co-processor”).
- 📠 In the smart-card context, what would be required to run (ideal) lattice-based cryptography?

³And DES!

Deploying cryptography

○○●○○

Deployment in general

Rings on RSA co-processors

○○○○○○○○○○○○○○

Implementation

○○○○○

Future directions

○○○

Ideal lattices

Definition (MLWE as used in Kyber)

Let $R = \mathbb{Z}[x]/(x^n + 1)$ where n is a power of 2, let $R_q = R/(q)$ for some $q \in \mathbb{Z}_+$.

Let R_q^k be a ring module of dimension k over R_q . Let χ be a probability distribution over \mathbb{Z}_q .

$$\vec{b} = A \stackrel{\$}{\leftarrow} R_q^{k \times k} \cdot \vec{s} \stackrel{\chi}{\leftarrow} R_q^k + \vec{e} \stackrel{\chi}{\leftarrow} R_q^k$$

🗄️ *Decision-MLWE*: distinguish (A, \vec{b}) from uniform

🗄️ *Search-MLWE*: recover \vec{s} from (A, \vec{b})

🗄️ *Note*: every row $\vec{b}_i = \sum_j A_{i,j} \cdot \vec{s}_j + \vec{e}_i$

Definition (Kyber CPA PKE component)

Simplified Kyber.CPA.Gen

- 1 $A \xleftarrow{\$} R_q^{k \times k}$
- 2 $(\vec{s}, \vec{e}) \xleftarrow{\chi} R_q^k \times R_q^k$
- 3 $\vec{t} \leftarrow \text{COMPRESS}_q(A\vec{s} + \vec{e})$
- 4 **return** $pk_{CPA} := (\vec{t}, A)$, $sk_{CPA} := \vec{s}$

Simplified Kyber.CPA.Dec

Input: $sk_{CPA} = \vec{s}$

Input: $c = (\vec{u}, \vec{v})$

- 1 $\vec{u} \leftarrow \text{DECOMPRESS}_q(\vec{u})$
- 2 $\vec{v} \leftarrow \text{DECOMPRESS}_q(\vec{v})$
- 3 **return** $\text{COMPRESS}_q(\vec{v} - \langle \vec{s}, \vec{u} \rangle)$

Simplified Kyber.CPA.Enc

Input: $pk_{CPA} = (\vec{t}, A)$

Input: $m \in \mathcal{M}$

- 1 $\vec{t} \leftarrow \text{DECOMPRESS}_q(\vec{t})$
- 2 $(\vec{r}, \vec{e}_1, \vec{e}_2) \xleftarrow{\chi} R_q^k \times R_q^k \times R_q$
- 3 $\vec{u} \leftarrow \text{COMPRESS}_q(A^T \vec{r} + \vec{e}_1)$
- 4 $\vec{v} \leftarrow \text{COMPRESS}_q(\langle \vec{t}, \vec{r} \rangle + \vec{e}_2 + \lceil \frac{q}{2} \rceil \cdot m)$
- 5 **return** $c := (\vec{u}, \vec{v})$

The CCA-secure Kyber768 KEM from the 1st round, is obtained by setting $n = 256$, $k = 3$, $q = 7681$ and using a FO-like transform.

🧮 The most expensive operation is computing $MULADD(a, b, c)$:

$$a(x) \cdot b(x) + c(x) \bmod (q, f(x)).$$

🧮 To reduce its cost, the \cdot is computed using the Number Theoretic Transform (NTT).

🧩 The most expensive operation is computing $MULADD(a, b, c)$:

$$a(x) \cdot b(x) + c(x) \bmod (q, f(x)).$$

🧩 To reduce its cost, the \cdot is computed using the Number Theoretic Transform (NTT).

🧩 In the embedded hardware setting, multiple designs for “RLWE co-processors” have been proposed⁴.

🧩 Yet, new hardware design means having to implement, test, certify, and deploy!

⁴E.g. [GFS⁺12] [PG12] [APS13] [PG14a] [PG14b] [PDG14] [RVM⁺14] [CMV⁺15] [POG15] [RRVV15] [LPO⁺17]...



- Our approach: we construct a flexible *MULADD* gadget by reusing the RSA co-processor on current smart-cards.
- We demonstrate it by implementing a variant of Kyber with competitive performance on the SLE 78 platform.
- Throughout this work we refer to the original NIST PQC's first round design/parameters of Kyber.

Arithmetic

Kronecker substitution

- 📄 Kronecker substitution (KS) is a classical technique in computational algebra for reducing polynomial arithmetic to large integer arithmetic [VZGG13, p. 245][Har09].

Kronecker substitution

-  Kronecker substitution (KS) is a classical technique in computational algebra for reducing polynomial arithmetic to large integer arithmetic [VZGG13, p. 245][Har09].
-  The fundamental idea behind this technique is that univariate polynomial and integer arithmetic are identical except for carry propagation in the latter.

$$a = x + 2$$

$$b = 3x + 4$$



$$a \cdot b = 3x^2 + 10x + 8$$

$$A = a(100) = 100 + 2$$

$$B = b(100) = 3 \cdot 100 + 4$$

$$\begin{aligned} A \cdot B &= 102 \cdot 304 = 31008 \\ &= 3 \cdot 100^2 + 10 \cdot 100 + 8 \end{aligned}$$

Kronecker substitution

-  Kronecker substitution (KS) is a classical technique in computational algebra for reducing polynomial arithmetic to large integer arithmetic [VZGG13, p. 245][Har09].
-  The fundamental idea behind this technique is that univariate polynomial and integer arithmetic are identical except for carry propagation in the latter.

$$a = x + 2$$


$$b = 3x + 4$$

$$a \cdot b = 3x^2 + 10x + 8$$

$$A = a(100) = 100 + 2$$

$$B = b(100) = 3 \cdot 100 + 4$$

$$\begin{aligned}
 A \cdot B &= 102 \cdot 304 = 31008 \\
 &= 3 \cdot 100^2 + 10 \cdot 100 + 8
 \end{aligned}$$

-  This works if we choose a large enough integer to evaluate a and b on. It also works for signed coefficients [Har09].

It also works when evaluating $a(x) \bmod f(x)$:

$$a = 3x^2 + 10x + 8$$

$$f = x^2 + 1$$

$$a \bmod f = 3x^2 + 10x + 8$$

$$- 3(x^2 + 1)$$

$$= 10x + 5$$


$$A = a(100) = 3 \cdot 100^2 + 10 \cdot 100 + 8$$

$$F = f(100) = 100^2 + 1$$

$$A \bmod F = 3 \cdot 100^2 + 10 \cdot 100 + 8$$

$$- 3(100^2 + 1)$$

$$= 1005 = 10 \cdot 100 + 5$$

 By combining the two properties, and choosing fixed representatives for coefficients in \mathbb{Z}_q , it is possible to compute

$$a(x) \cdot b(x) + c(x) \bmod (q, f(x))$$

by

$$a(t) \cdot b(t) + c(t) \bmod f(t)$$

where $t \in \mathbb{Z}$ is large enough.

- By combining the two properties, and choosing fixed representatives for coefficients in \mathbb{Z}_q , it is possible to compute

$$a(x) \cdot b(x) + c(x) \bmod (q, f(x))$$

by

$$a(t) \cdot b(t) + c(t) \bmod f(t)$$

where $t \in \mathbb{Z}$ is large enough.

- Since these are all integers, we can use our RSA co-processor to compute in $\mathbb{Z}_{f(t)}$!

- Say KS requires us to work with $t = 2^{2^\ell}$, but we needed more compact packing (foreshadowing intensifies...)
- [Har09] introduces a variant of Kronecker substitution that further shortens t .

Kronecker substitution

“KS2”: say we need $t = 2^{2^\ell}$. Let

$$c^{(+)} := c(2^\ell) = a(2^\ell) \cdot b(2^\ell) = \sum_{[i]_2=0} c_i 2^{i\ell} + \sum_{[i]_2=1} c_i 2^{i\ell}$$

$$c^{(-)} := c(-2^\ell) = a(-2^\ell) \cdot b(-2^\ell) = \sum_{[i]_2=0} c_i 2^{i\ell} - \sum_{[i]_2=1} c_i 2^{i\ell}$$

Kronecker substitution

“KS2”: say we need $t = 2^{2^\ell}$. Let

$$c^{(+)} := c(2^\ell) = a(2^\ell) \cdot b(2^\ell) = \sum_{[i]_2=0} c_i 2^{i\ell} + \sum_{[i]_2=1} c_i 2^{i\ell}$$

$$c^{(-)} := c(-2^\ell) = a(-2^\ell) \cdot b(-2^\ell) = \sum_{[i]_2=0} c_i 2^{i\ell} - \sum_{[i]_2=1} c_i 2^{i\ell}$$

Then, we can recover the even coefficients of $c(x)$ from

$$c^{(+)} + c^{(-)} = c(2^\ell) + c(-2^\ell) = 2 \sum_{[i]_2=0} c_i 2^{i\ell}$$

and the odd coefficients from

$$c^{(+)} - c^{(-)} = c(2^\ell) - c(-2^\ell) = 2 \cdot 2^\ell \sum_{[i]_2=1} c_i 2^{(i-1)\ell}$$

since the sum and the difference cancel out either the even or the odd powers. KS2 is also compatible with arithmetic modulo $f = x^n + 1$, when n is even.

- How should we chose $t \in \mathbb{Z}$?
- In [AHH⁺18], we provide a tight lower bound such that the computation works without errors by carry.

How should we choose $t \in \mathbb{Z}$?

In [AHH⁺18], we provide a tight lower bound such that the computation works without errors by carry.

Lemma

Let $a, b, c \in R_q$ such that $\|a\|_\infty \leq \alpha$, $\|b\|_\infty \leq \beta$, $\|c\|_\infty \leq \gamma$. Let

$$d := \sum_{i=0}^{n-1} d_i x^i \equiv a \cdot b + c \pmod{f}$$

with $\|d\|_\infty \leq \delta(\alpha, \beta, \gamma, n, f)$, where f is monic of degree n such that $f(2^\ell) > 2^{n\ell} - 1$. Let $\varphi := \max_{i < n} |f_i|$, and let $\ell > \log_2(\delta + \varphi) + 1$ be an integer. Then the above tricks work for any integer $t \geq 2^\ell$.

Let's see, for Kyber768 ($k = 3, n = 256, q = 7681, \eta = 4$)


$$\ell > \log_2 \left(kn \left\lfloor \frac{q}{2} \right\rfloor \eta + \eta + 1 \right) + 1 \approx 24.5 \implies \ell = 25.$$

This means having $\log_2 f(t) = \log_2 f(2^\ell) > \ell \cdot n = 6400$.

Problem: our RSA multiplier computes $x \cdot y \bmod z$ where $\log x, \log y, \log z < 2200$.

Splitting rings

- KS alone won't suffice.
- We can interpolate between full polynomial multiplication and KS.
- The idea is similar to Schönhage [Sch77] or Nussbaumer [Nus80].

 Say we have

$$a = a_0 + a_1 x + \cdots + a_4 x^4 + a_5 x^5$$

$$f = x^6 + 1 \quad (\text{non-power-of-two for example's sake}).$$

🧩 Say we have

$$a = a_0 + a_1 x + \cdots + a_4 x^4 + a_5 x^5$$


$$f = x^6 + 1 \quad (\text{non-power-of-two for example's sake}).$$

🧩 Add a dummy variable $y = x^2$;

 Say we have

$$a = a_0 + a_1 x + \cdots + a_4 x^4 + a_5 x^5$$

$$f = x^6 + 1 \quad (\text{non-power-of-two for example's sake}).$$

 Add a dummy variable $y = x^2$; then

$$\begin{aligned} a &\equiv (a_0 + a_2 y + a_4 y^2) + (a_1 + a_3 y + a_5 y^2) x \pmod{(y - x^2)} \\ &= a^{(0)}(y) + a^{(1)}(y) x. \end{aligned}$$

🗄️ Say we have

$$a = a_0 + a_1 x + \cdots + a_4 x^4 + a_5 x^5$$


$$f = x^6 + 1 \quad (\text{non-power-of-two for example's sake}).$$

🗄️ Add a dummy variable $y = x^2$; then

$$\begin{aligned} a &\equiv (a_0 + a_2 y + a_4 y^2) + (a_1 + a_3 y + a_5 y^2) x \pmod{(y - x^2)} \\ &= a^{(0)}(y) + a^{(1)}(y) x. \end{aligned}$$

🗄️ Similarly, say we have $b \equiv b^{(0)}(y) + b^{(1)}(y) x \pmod{(y - x^2)}$.

Splitting rings

 Compute $a \cdot b \bmod f \equiv (a \cdot b \bmod y^2 + 1) \bmod x^4 + 1$.

Splitting rings

🗄️ Compute $a \cdot b \bmod f \equiv (a \cdot b \bmod y^2 + 1) \bmod x^4 + 1$.

🗄️ The inner operation is

$$a \cdot b \bmod y^2 + 1 = a^{(0)} b^{(0)} + a^{(1)} b^{(1)} x^2 \\ + (a^{(1)} b^{(0)} + a^{(0)} b^{(1)}) x \bmod y^2 + 1$$

where each $a^{(i)} b^{(j)} \bmod y^2 + 1$ can be computed using KS, but packing polynomials of smaller degree.

🗄️ Hence, a smaller ℓ and multiplier are require wrt the naive approach.

Splitting rings

Compute $a \cdot b \bmod f \equiv (a \cdot b \bmod y^2 + 1) \bmod x^4 + 1$.

The inner operation is

$$\begin{aligned} a \cdot b \bmod y^2 + 1 &= a^{(0)} b^{(0)} + a^{(1)} b^{(1)} x^2 \\ &\quad + (a^{(1)} b^{(0)} + a^{(0)} b^{(1)}) x \bmod y^2 + 1 \end{aligned}$$

where each $a^{(i)} b^{(j)} \bmod y^2 + 1$ can be computed using KS, but packing polynomials of smaller degree.

Hence, a smaller ℓ and multiplier are required wrt the naive approach.

This results in a polynomial in x of degree < 4 to reduce mod f , which can be done on the CPU.

Splitting rings

🗄️ Compute $a \cdot b \bmod f \equiv (a \cdot b \bmod y^2 + 1) \bmod x^4 + 1$.

🗄️ The inner operation is

$$a \cdot b \bmod y^2 + 1 = a^{(0)} b^{(0)} + a^{(1)} b^{(1)} x^2 \\ + (a^{(1)} b^{(0)} + a^{(0)} b^{(1)}) x \bmod y^2 + 1$$

where each $a^{(i)} b^{(j)} \bmod y^2 + 1$ can be computed using KS, but packing polynomials of smaller degree.

🗄️ Hence, a smaller ℓ and multiplier are required wrt the naive approach.

🗄️ This results in a polynomial in x of degree < 4 to reduce mod f , which can be done on the CPU.

🗄️ This technique enables us to compute the Kyber768 MULADD operation using e.g. polynomials of y -degree < 64 , x -degree < 4 , and $\ell \geq 25$ (we choose $\ell = 32$). Round 2 Kyber may even fit in $\ell = 24$.





Karatsuba multiplication

- One more trick: since we are now multiplying low-degree polynomials in x , we can use Karatsuba-like formulae.
- In its simplest form, the algorithm computes $(a + b \cdot x) \cdot (c + d \cdot x)$ in $\mathbb{Z}[x]$ by computing the products $t_0 = a \cdot c$, $t_1 = b \cdot d$ and $t_2 = (a + b) \cdot (c + d)$ and outputting $t_0 + (t_2 - t_0 - t_1) \cdot x + t_1 x^2$.

Karatsuba multiplication

- One more trick: since we are now multiplying low-degree polynomials in x , we can use Karatsuba-like formulae.
- In its simplest form, the algorithm computes $(a + b \cdot x) \cdot (c + d \cdot x)$ in $\mathbb{Z}[x]$ by computing the products $t_0 = a \cdot c$, $t_1 = b \cdot d$ and $t_2 = (a + b) \cdot (c + d)$ and outputting $t_0 + (t_2 - t_0 - t_1) \cdot x + t_1 x^2$.
- This can be done recursively, to obtain a complexity of $3^{\lceil \log_2 L \rceil}$ coefficient multiplications for degree $L - 1$ polynomials, versus schoolbook multiplication using L^2 multiplications.

Karatsuba multiplication

-  One more trick: since we are now multiplying low-degree polynomials in x , we can use Karatsuba-like formulae.
-  In its simplest form, the algorithm computes $(a + b \cdot x) \cdot (c + d \cdot x)$ in $\mathbb{Z}[x]$ by computing the products $t_0 = a \cdot c$, $t_1 = b \cdot d$ and $t_2 = (a + b) \cdot (c + d)$ and outputting $t_0 + (t_2 - t_0 - t_1) \cdot x + t_2 x^2$.
-  This can be done recursively, to obtain a complexity of $3^{\lceil \log_2 L \rceil}$ coefficient multiplications for degree $L - 1$ polynomials, versus schoolbook multiplication using L^2 multiplications.
-  One can also halve the degrees using KS2, ending up with no need for Karatsuba.


Deploying cryptography
○○○○○○○

Rings on RSA co-processors
○○○○○○○○○○○○○○

Implementation
●○○○○

Future directions
○○○

Implementation

 After all this work, we have a MULADD gadget running on an RSA co-processor. Is it worth it in practice?

- After all this work, we have a MULADD gadget running on an RSA co-processor. Is it worth it in practice?
- Kyber makes use of SHAKE-128 as XOF, SHAKE-256 as PRF, and SHA3 as hash function for the CCA transform.
- The SLE 78 has no Keccak-f co-processor, and software implementations are way too slow.
- We circumvent this problem by defining an AES-based XOF and PRF, and use SHA256 for the CCA transform's G and H .
- A similar variant was introduced in NIST PQC's second round Kyber revision as "Kyber-90s".

Table: Performance of our work on the SLE 78 target device in clock cycles.

Scheme	Cycles
KYBER.CPA.IMP.GEN (HW-AES: PRF/XOF)	3,625,718
KYBER.CPA.IMP.ENC (HW-AES: PRF/XOF)	4,747,291
KYBER.CPA.IMP.DEC	1,420,367
KYBER.CCA.IMP.GEN (HW-AES: PRF/XOF; SW-SHA3: <i>H</i> ; KS2)	14,512,691
KYBER.CCA.IMP.ENC (HW-AES: PRF/XOF; SW-SHA3: <i>G, H</i> ; KS2)	18,051,747
KYBER.CCA.IMP.DEC (HW-AES: PRF/XOF; SW-SHA3: <i>G, H</i> ; KS2)	19,702,139
KYBER.CCA.IMP.GEN (HW-AES: PRF/XOF; HW-SHA-256: <i>H</i> ; KS2)	3,980,517
KYBER.CCA.IMP.ENC (HW-AES: PRF/XOF; HW-SHA-256: <i>G, H</i> ; KS2)	5,117,996
KYBER.CCA.IMP.DEC (HW-AES: PRF/XOF; HW-SHA-256: <i>G, H</i> ; KS2)	6,632,704

Table: Comparison of our work with other PKE or KEM schemes on SLE 78.

Scheme	Target	Gen	Enc	Dec
Kyber768 ^a (CPA; our work)	SLE 78	3,625,718	4,747,291	1,420,367
Kyber768 ^b (CCA; our work)	SLE 78	3,980,517	5,117,996	6,632,704
RSA-2048 ^c	SLE 78	-	≈ 300,000	≈ 21,200,000
RSA-2048 (CRT) ^d	SLE 78	-	≈ 300,000	≈ 6,000,000
Kyber768 (CPA+NTT) ^e	SLE 78	≈ 10,000,000	≈ 14,600,000	≈ 5,400,000
NewHope1024 ^f	SLE 78	≈ 14,700,000	≈ 31,800,000	≈ 15,200,000

^a CPA-secure Kyber variant using the AES co-processor to implement PRF/XOF and KS2 on SLE 78 @ 50 MHz.

^b CCA-secure Kyber variant using the AES co-processor to implement PRF/XOF, the SHA-256 co-processor to implement G and H and KS2 on SLE 78 @ 50 MHz.

^c RSA-2048 encryption with short exponent and decryption without CRT and with countermeasures on SLE 78 @ 50 MHz. Extrapolation based on data-sheet.

^d RSA-2048 decryption with short exponent and decryption with CRT and countermeasures on SLE 78 @ 50 MHz. Extrapolation based on data-sheet.

^e Extrapolation of cycle counts of CPA-secure Kyber768 based on our implementation assuming usage of the AES co-processor to implement PRF/XOF and a software implementation of the NTT with 997,691 cycles for an NTT on SLE 78 @ 50 MHz.

^f Reference implementation of constant time ephemeral NewHope key exchange ($n = 1024$) [ADPS16] modified to use the AES co-processor as PRNG on SLE 78 @ 50 MHz.

Deploying cryptography
○○○○○○○

Rings on RSA co-processors
○○○○○○○○○○○○○○

Implementation
○○○○●

Future directions
○○○

Future directions

Investigate other schemes:

- 🐻 ThreeBears [Ham17] is a NIST proposal designed with a similar idea of doing lattice-based cryptography over the integers. However, integer sizes too large for direct handling with *our*⁵ co-processor (RSA 8192 anyone?).

⁵And by “our” I mean Infineon’s.

Investigate other schemes:

- ThreeBears [Ham17] is a NIST proposal designed with a similar idea of doing lattice-based cryptography over the integers. However, integer sizes too large for direct handling with *our*⁵ co-processor (RSA 8192 anyone?).
- Try implementing an MLWE-based scheme that is parameterised with a power-of-two modulus q , e.g. SABER [DKRV17].

⁵And by “our” I mean Infineon’s.

Investigate other schemes:

- ThreeBears [Ham17] is a NIST proposal designed with a similar idea of doing lattice-based cryptography over the integers. However, integer sizes too large for direct handling with *our*⁵ co-processor (RSA 8192 anyone?).
- Try implementing an MLWE-based scheme that is parameterised with a power-of-two modulus q , e.g. SABER [DKRV17].
- Try designing a scheme with parameters such that each packed polynomial fits directly into a co-processor register (prime cyclotomic? Kyber with smaller non-NTT-friendly q ?).

⁵And by “our” I mean Infineon’s.

Investigate other schemes:

- 🧩 ThreeBears [Ham17] is a NIST proposal designed with a similar idea of doing lattice-based cryptography over the integers. However, integer sizes too large for direct handling with *our*⁵ co-processor (RSA 8192 anyone?).
- 🧩 Try implementing an MLWE-based scheme that is parameterised with a power-of-two modulus q , e.g. SABER [DKRV17].
- 🧩 Try designing a scheme with parameters such that each packed polynomial fits directly into a co-processor register (prime cyclotomic? Kyber with smaller non-NTT-friendly q ?).
- 🧩 Try implementing a signature scheme, e.g. Dilithium.

⁵And by “our” I mean Infineon’s.

Final idea:


- 🧩 LWE-based CPA schemes tolerate some small level of noise added to the ciphertext.
- 🧩 Maybe we can choose ℓ smaller than what our correctness lower bound requires.
- 🧩 We could introduce carry-over errors when computing


$$a \cdot b + c \pmod{f}$$


- 🧩 If we can bound the error norm, we may still get correct decryption, with smaller packed polynomials.

Thank you

You can find:

 the paper @ <https://ia.cr/2018/425>

 the code @
<https://github.com/fvirdia/lwe-on-rsa-copro>

 me @ <https://fundamental.domains>



Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe.
Post-quantum key exchange - A new hope.

In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.



Martin R. Albrecht, Christian Hanser, Andrea Hoeller, Thomas Pöppelmann, Fernando Virdia, and Andreas Wallner.

Implementing RLWE-based schemes using an RSA co-processor.

IACR TCHES, 2019(1):169–208, 2018.

<https://tches.iacr.org/index.php/TCHES/article/view/7338>.



A. Aysu, C. Patterson, and P. Schaumont.

Low-cost and area-efficient fpga implementations of lattice-based cryptography.

In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 81–86, June 2013.



Lejla Batina and Matthew Robshaw, editors.

CHES 2014, volume 8731 of *LNCS*. Springer, Heidelberg, September 2014.



D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede.

High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems.

IEEE Transactions on Circuits and Systems I: Regular Papers, 62(1):157–166, Jan 2015.



Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren.

Saber.

Technical report, National Institute of Standards and Technology, 2017.
available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>.



Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann, and Sorin A. Huss.

On the design of hardware building blocks for modern lattice-based encryption schemes.

In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 512–529. Springer, Heidelberg, September 2012.



Mike Hamburg.

Three bears.

Technical report, National Institute of Standards and Technology, 2017.
available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>.



David Harvey.

Faster polynomial multiplication via multipoint kronecker substitution.

J. Symb. Comput., 44(10):1502–1510, 2009.



Zhe Liu, Thomas Pöppelmann, Tobias Oder, Hwajeong Seo, Sujoy Sinha Roy, Tim Güneysu, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede.

Towards practical lattice-based public-key encryption on reconfigurable hardware.

In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 68–85. Springer, Heidelberg, August 2014.



T. Pöppelmann and T. Güneysu.

Area optimization of lightweight lattice-based encryption on reconfigurable hardware.

In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2796–2799, June 2014.



Thomas Pöppelmann, Tobias Oder, and Tim Güneysu.

High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers.

In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, Heidelberg, August 2015.



Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-LWE implementation.

In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 683–702. Springer, Heidelberg, September 2015.



Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede.

Compact ring-LWE cryptoprocessor.

In Batina and Robshaw [BR14], pages 371–391.



Arnold Schönhage.

Schnelle multiplikation von polynomen über körpern der charakteristik 2.

Acta Informatica, 7(4):395–398, Dec 1977.



Peter W. Shor.

Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.

SIAM J. Comput., 26(5):1484–1509, October 1997.



Joachim Von Zur Gathen and Jürgen Gerhard.

Modern computer algebra.

Cambridge university press, 2013.