Implementing RLWE-based Schemes Using an RSA Co-Processor

Martin R. Albrecht[1], Christian Hanser[2], Andrea Hoeller[2], Thomas Pöppelmann[3], Fernando Virdia[1], Andreas Wallner[2]

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

[2]Infineon Technologies Austria AG

[3]Infineon Technologies AG, Germany

23 January 2019
Lattice Coding & Crypto Meeting
London

Overview

- Prelude
  - Post-quantum cryptography
- Deploying cryptography
  - Deployment in general
  - Lattice-based cryptography
- Ring arithmetic on RSA co-processors
  - Kronecker substitution
  - Splitting rings
  - Karatsuba multiplication
- Implementation
- Future directions

Prelude

- [Sho97] introduces a fast[1] order-finding quantum algorithm that allows factoring and computing discrete logs in Abelian groups.
- Since then, there has been a growing effort to develop new public-key encryption and signature algorithms that can resist cryptanalysis using large-scale general quantum computers.

[1] Let's not go there.

- [Sho97] introduces a fast[1] order-finding quantum algorithm that allows factoring and computing discrete logs in Abelian groups.
- Since then, there has been a growing effort to develop new 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.

[1] Let's not go there.

Deploying cryptography

- In practice, cryptographic schemes have two crucial requirements[2]: 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.

[2] Other than being secure in some appropriate model!

- In practice, cryptographic schemes have two crucial requirements[2]: 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.
- However, implementations of quantum-safe schemes are also required in constrained (often embedded) environments such as microcontrollers or smart cards.

[2] Other than being secure in some appropriate model!

- For example, smart-cards provide low-power 16-bit and 32-bit CPU and small amounts of RAM.

[3] And DES!

- For example, smart-cards provide low-power 16-bit and 32-bit CPU 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 78CLUFX5000 Infineon chip card provides:
  - 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
  - AES and SHA256 co-processors[3],
  - $\mathbb{Z}_N$ adder and multiplier for $\log_2 N = 2200$ ("the RSA co-processor").

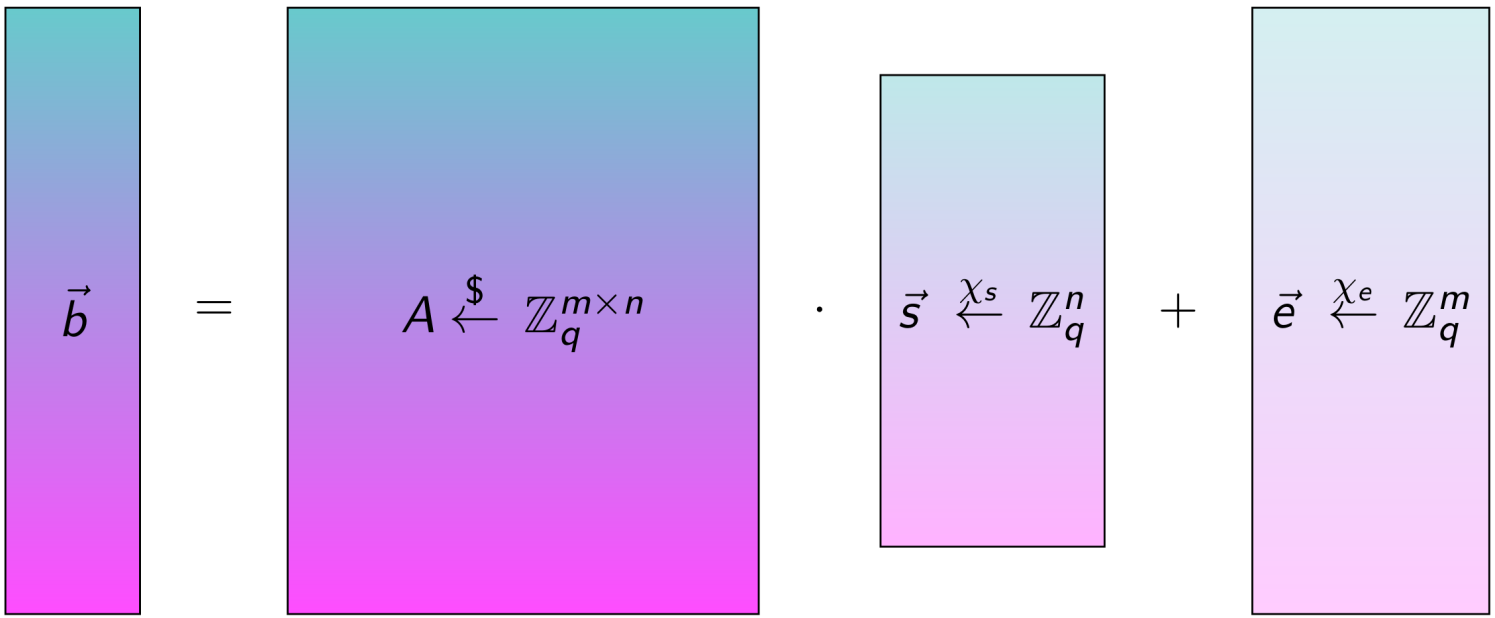
[3] And DES!

- For example, smart-cards provide low-power 16-bit and 32-bit CPU 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 78CLUFX5000 Infineon chip card provides:
  - 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
  - AES and SHA256 co-processors[3],
  - $\mathbb{Z}_N$ adder and multiplier for $\log_2 N = 2200$ ("the RSA co-processor").
- In the smart-card context, what would be required to run lattice-based cryptography?

[3] And DES!

Definition (LWE)

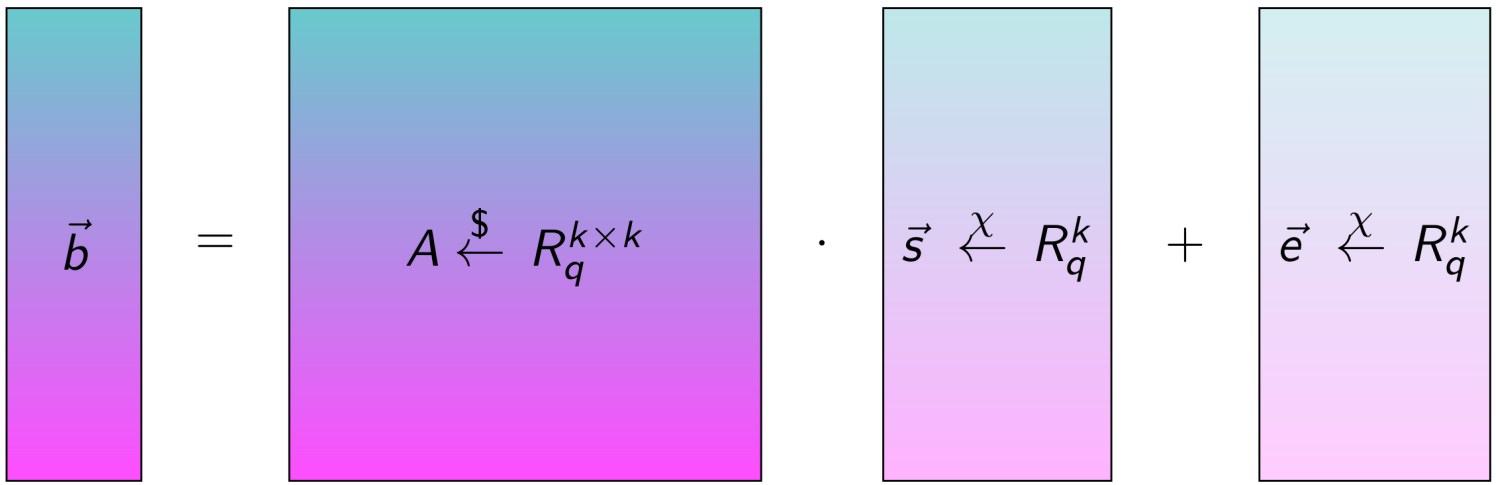
For $q, n, m \in \mathbb{Z}_+$ with $m = O(n)$, χ_s, χ_e probability distributions over $\mathbb{Z}_q$,



- *Decision-LWE*: distinguish $(A, \vec{b})$ from uniform
- *Search-LWE*: recover $\vec{s}$ from $(A, \vec{b})$

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 $\mathbb{Z}_q$.



- *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 $\vec{t} \leftarrow \textsc{Compress}_q(A\vec{s} + \vec{e})$
4 **return**
$pk_{CPA} := (\vec{t}, A)$, $sk_{CPA} := \vec{s}$

Simplified Kyber.CPA.Enc

Input: $pk_{CPA} = (\vec{t}, A)$
Input: $m \in \mathcal{M}$
1 $\vec{t} \leftarrow \textsc{Decompress}_q(\vec{t})$
2 $(\vec{r}, \vec{e}_1, e_2) \xleftarrow{\chi} R_q^k \times R_q^k \times R_q$
3 $\vec{u} \leftarrow \textsc{Compress}_q(A^T\vec{r} + \vec{e}_1)$
4 $v \leftarrow \textsc{Compress}_q(\langle \vec{t}, \vec{r} \rangle + e_2 + \lceil \frac{q}{2} \rfloor \cdot m)$
5 **return** $c := (\vec{u}, v)$

Simplified Kyber.CPA.Dec

Input: $sk_{CPA} = \vec{s}$
Input: $c = (\vec{u}, v)$
1 $\vec{u} \leftarrow \textsc{Decompress}_q(\vec{u})$
2 $v \leftarrow \textsc{Decompress}_q(v)$
3 **return** $\textsc{Compress}_q(v - \langle \vec{s}, \vec{u} \rangle)$

The CCA-secure Kyber768 KEM 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 proposed[4].

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

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

Ring arithmetic on RSA co-processors

- 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.

- 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.

Kronecker substitution

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

Kronecker substitution

- Kronecker substitution 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.

$$\begin{aligned} a &= x + 2 \\ b &= 3x + 4 \\ a \cdot b &= 3x^2 + 10x + 8 \end{aligned}$$

$$\begin{aligned} A &= a(100) = 100 + 2 \\ B &= b(100) = 3 \cdot 100 + 4 \\ A \cdot B &= 102 \cdot 304 = 31008 \\ &= 3 \cdot 100^2 + 10 \cdot 100 + 8 \end{aligned}$$

Kronecker substitution

- Kronecker substitution 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.

$$a = x + 2$$
$$b = 3x + 4$$
$$a \cdot b = 3x^2 + 10x + 8$$

$$A = a(100) = 100 + 2$$
$$B = b(100) = 3 \cdot 100 + 4$$
$$A \cdot B = 102 \cdot 304 = 31008$$
$$= 3 \cdot 100^2 + 10 \cdot 100 + 8$$

- 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 $a(x)$ mod $f(x)$:

$$\begin{aligned} a &= 3x^2 + 10x + 8 \\ f &= x^2 + 1 \\ a \bmod f &= 3x^2 + 10x + 8 \\ &\quad - 3(x^2 \qquad + 1) \\ &= \qquad 10x + 5 \end{aligned}$$

$$\begin{aligned} A &= a(100) = 3 \cdot 100^2 + 10 \cdot 100 + 8 \\ F &= f(100) = 100^2 + 1 \\ A \bmod F &= 3 \cdot 100^2 + 10 \cdot 100 + 8 \\ &\quad - 3(100^2 \qquad + 1) \\ &= 1005 = 10 \cdot 100 + 5 \end{aligned}$$

- 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 $\mathbb{Z}_{f(t)}$!

- The particular variant we use furthermore shortens t.

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

Lemma
Let $a, b, c \in \mathbb{Z}[x]$ such that $a = \sum_{i=0}^{n-1} a_i x^i$, $b = \sum_{i=0}^{n-1} b_i x^i$, $c = \sum_{i=0}^{n-1} c_i x^i$ with $a_i \in \{-\alpha, \ldots, \alpha\}$, $b_i \in \{-\beta, \ldots, \beta\}$, and $c_i \in \{-\gamma, \ldots, \gamma\}$. Let

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

with $d_i \in \{-\delta, \ldots, \delta\}$, where $\delta > 0$ depends on $\alpha, \beta, \gamma, n, f$ and 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.

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].

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].
- Let's abuse notation.

Say we have

$$a = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3$$
$$b = b_0 + b_1\,x + b_2\,x^2 + b_3\,x^3$$
$$f = x^4 + 1$$

and we want to compute $a \cdot b \bmod f$.

Let $y = x^2$; then

$$a = a^{(0)} + a^{(1)}\,x$$

where

$$a^{(0)} = a_0 + a_2\,y \quad \text{and} \quad a^{(1)} = a_1 + a_3\,y,$$

and similarly for b.
Then, computing $a \cdot b \bmod f \equiv (a \cdot b \bmod y^2 + 1) \bmod x^4 + 1$.

- The inner operation is

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

where each $a^{(i)}\, b^{(j)} \bmod y^2 + 1$ can be computed using KS, with a smaller ℓ than the original operation would require.

- 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, with a smaller ℓ than the original operation would require.

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

- While in this small example there is no gain, this technique enables us to compute the Kyber768 MULADD operation using e.g. polynomials of y-degree < 64, x-degree < 4, and $\ell > 25$ (we choose $\ell = 32$).

Karatsuba multiplication

- One more trick: since we are now multiplying low-degree polynomials in x, we can use Karatsuba-like formulae.

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 $t_0 + (t_2 - t_0 - t_1) \cdot x + t_2 x^2$.

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.

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.

- 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.

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: H)	14,512,691
Kyber.CCA.Imp.Enc (HW-AES: PRF/XOF; SW-SHA3: G, H)	18,051,747
Kyber.CCA.Imp.Dec (HW-AES: PRF/XOF; SW-SHA3: G, H)	19,702,139
Kyber.CCA.Imp.Gen (HW-AES: PRF/XOF; HW-SHA-256: H)	3,980,517
Kyber.CCA.Imp.Enc (HW-AES: PRF/XOF; HW-SHA-256: G, H)	5,117,996
Kyber.CCA.Imp.Dec (HW-AES: PRF/XOF; HW-SHA-256: G, H)	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	-	$\approx$ 300,000	$\approx$ 21,200,000
RSA-2048 (CRT)[d]	SLE 78	-	$\approx$ 300,000	$\approx$ 6,000,000
Kyber768 (CPA+NTT)[e]	SLE 78	$\approx$ 10,000,000	$\approx$ 14,600,000	$\approx$ 5,400,000
NewHope1024[f]	SLE 78	$\approx$ 14,700,000	$\approx$ 31,800,000	$\approx$ 15,200,000

[a] CPA-secure Kyber variant using the AES co-processor to implement $\mathrm{PRF}/\mathrm{XOF}$ and KS2 on SLE 78 @ 50 MHz.

[b] CCA-secure Kyber variant using the AES co-processor to implement $\mathrm{PRF}/\mathrm{XOF}$, 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. Extrapoliation based on data-sheet.

[d] RSA-2048 decryption with short exponent and decryption with CRT and countermeasures on SLE 78 @ 50 MHz. Extrapoliation 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 $\mathrm{PRF}/\mathrm{XOF}$ 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.

Future directions

Investigate other schemes:

- ThreeBears [Ham17] or Mersenne-75683917 [AJPS17] are NIST proposals designed with a similar idea of doing lattice-based cryptography over the integers. However, they use integer sizes too large for direct handling with our co-processor.
- 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.

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

$$a \cdot b + c \mod f.$$

- 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`

[Har09] introduces a KS variant working as follows. Assume we are computing $a \cdot b$ using $t = 2^{2\ell}$. Let

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

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

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. The KS2 algorithm is compatible with arithmetic modulo $f = x^n + 1$, when n is even.


Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe.
Post-quantum key exchange - A new hope.
In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16*, pages 327–343. USENIX Association, 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.


Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Mikos Santha.
Mersenne-756839.
Technical report, National Institute of Standards and Technology, 2017.
available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.


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.
High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers.
ACM Trans. Embedded Comput. Syst., 16(4):117:1–117:24, 2017.


National Institute of Standards and Technology.
Submission requirements and evaluation criteria for the Post-Quantum Cryptography standardization process.
`http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`, December 2016.


H. Nussbaumer.
Fast polynomial transform algorithms for digital convolution.
IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(2):205–215, Apr 1980.


Thomas Pöppelmann, Léo Ducas, and Tim Güneysu.
Enhanced lattice-based signatures on reconfigurable hardware.
In Batina and Robshaw [BR14], pages 353–370.


Thomas Pöppelmann and Tim Güneysu.
Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware.

In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 139–158. Springer, Heidelberg, October 2012.

Thomas Pöppelmann and Tim Güneysu.
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.