Implementing RLWE-based Schemes Using an RSA Co-Processor

Martin R. Albrecht\textsuperscript{1}, Christian Hanser\textsuperscript{2}, Andrea Hoeller\textsuperscript{2}, Thomas Pöppelmann\textsuperscript{3}, Fernando Virdia\textsuperscript{1}, Andreas Wallner\textsuperscript{2}

\textsuperscript{1}Information Security Group, Royal Holloway, University of London, UK
\textsuperscript{2}Infineon Technologies Austria AG
\textsuperscript{3}Infineon Technologies AG, Germany

August 26, 2019
CHES 2019
Atlanta, GA
Overview

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

[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 primitives that can resist cryptanalysis using large-scale general quantum computers.

Many of the schemes proposed to NIST for standardisation are based on problems defined over polynomial rings, such as the RLWE problem.

\(^1\)Let’s not go there.
In practice, cryptographic schemes have two crucial requirements\textsuperscript{2}: high performance and ease of deployment.

Optimised implementations are an active area of research.

\textsuperscript{2}Other than being secure in some appropriate model!
In practice, cryptographic schemes have two crucial requirements\textsuperscript{2}: high performance and ease of deployment.

Optimised implementations are an active area of research.

As part of the NIST process, designers were required to provide 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 \textit{smart cards}.
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 78CLUFX5000 Infineon chip card provides:

- 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
- AES and SHA256 co-processors (and DES!),
- $\mathbb{Z}_N$ adder and multiplier for $\log_2 N = 2200$ (“the RSA co-processor”).
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 78CLUFX5000 Infineon chip card provides:

- 16-bit CPU @ 50 MHz, 16 Kbyte RAM, 500 Kbyte NVM,
- AES and SHA256 co-processors (and DES!),
- $\mathbb{Z}_N$ adder and multiplier for $\log_2 N = 2200$ (“the RSA co-processor”).

In this smart-card context, what would be required to run (ideal) lattice-based cryptography?
The most expensive operation in RLWE-based schemes is computing \( MULADD(a, b, c) \):

\[
a(x) \cdot b(x) + c(x) \mod (q, f(x)).
\]

To reduce its cost, the \( \cdot \) is often computed using the Number Theoretic Transform (NTT).
The most expensive operation in RLWE-based schemes is computing $MULADD(a, b, c)$:

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

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

In the embedded hardware setting, multiple designs for RLWE co-processors have been proposed$^3$.

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

---

$^3$E.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.
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.

\[
\begin{align*}
a &= x + 2 \\
b &= 3x + 4 \\
ab &= 3x^2 + 10x + 8
\end{align*}
\]

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

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

\[
\begin{align*}
    a &= 3x^2 + 10x + 8 \\
    f &= x^2 + 1 \\
    a \mod f &= 3x^2 + 10x + 8 \\
        &\quad - 3(x^2 + 1) \\
        &= 10x + 5
\end{align*}
\]

\[
\begin{align*}
    A &= a(100) = 3 \cdot 100^2 + 10 \cdot 100 + 8 \\
    F &= f(100) = 100^2 + 1 \\
    A \mod F &= 3 \cdot 100^2 + 10 \cdot 100 + 8 \\
        &\quad - 3(100^2 + 1) \\
        &= 1005 = 10 \cdot 100 + 5
\end{align*}
\]
By combining the two properties, and choosing fixed representatives for coefficients in $\mathbb{Z}_q$, it is possible to compute

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

by

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

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

Since these are all integers, we can use our RSA co-processor to compute in $\mathbb{Z}_{f(t)}$!
How should we chose $t = 2^\ell \in \mathbb{Z}$? In [AHH+18], we provide a tight lower bound for correctness.
How should we chose $t = 2^\ell \in \mathbb{Z}$? In [AHH$^+18$], we provide a tight lower bound for correctness.

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 $x \cdot y \mod z$ 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].
## 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].

- The idea: \[ a_0 + a_1 x + \cdots + a_4 x^4 + a_5 x^5 = (a_0 + a_2 y + a_4 y^2) + (a_1 + a_3 y + a_5 y^2) \times \mod (y - x^2). \]

- This technique enables us to compute the Kyber768 MULADD operation by combining Karatsuba-like multiplication of, say, degree 4 in \( x \) with KS for polynomials of degree 64 in \( y \), using \( \ell > 25 \) (we choose \( \ell = 32 \)).
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?

Round 1 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?

Round 1 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: Comparison of our work with other PKE or KEM schemes on SLE 78.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Target</th>
<th>Gen</th>
<th>Enc</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber768&lt;sup&gt;a&lt;/sup&gt; (CPA; our work)</td>
<td>SLE 78</td>
<td>3,625,718</td>
<td>4,747,291</td>
<td>1,420,367</td>
</tr>
<tr>
<td>Kyber768&lt;sup&gt;b&lt;/sup&gt; (CCA; our work)</td>
<td>SLE 78</td>
<td>3,980,517</td>
<td>5,117,996</td>
<td>6,632,704</td>
</tr>
<tr>
<td>RSA-2048&lt;sup&gt;c&lt;/sup&gt;</td>
<td>SLE 78</td>
<td>-</td>
<td>(\approx 300,000)</td>
<td>(\approx 21,200,000)</td>
</tr>
<tr>
<td>RSA-2048 (CRT)&lt;sup&gt;d&lt;/sup&gt;</td>
<td>SLE 78</td>
<td>-</td>
<td>(\approx 300,000)</td>
<td>(\approx 6,000,000)</td>
</tr>
<tr>
<td>Kyber768 (CPA+NTT)&lt;sup&gt;e&lt;/sup&gt;</td>
<td>SLE 78</td>
<td>(\approx 10,000,000)</td>
<td>(\approx 14,600,000)</td>
<td>(\approx 5,400,000)</td>
</tr>
<tr>
<td>NewHope1024&lt;sup&gt;f&lt;/sup&gt;</td>
<td>SLE 78</td>
<td>(\approx 14,700,000)</td>
<td>(\approx 31,800,000)</td>
<td>(\approx 15,200,000)</td>
</tr>
</tbody>
</table>

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

<sup>b</sup> 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.

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

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

<sup>e</sup> 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.

<sup>f</sup> 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.
Investigate other schemes:

- ThreeBears [Ham17] (uses only integers, but they are too long for the SLE 78 co-processor) or SABER [DKRV17] (similar design, power-of-two $q$).

- 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 $\ell$ 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
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber.CPA.Imp.Gen \ (HW-AES: PRF/XOF)</td>
<td>3,625,718</td>
</tr>
<tr>
<td>Kyber.CPA.Imp.Enc \ (HW-AES: PRF/XOF)</td>
<td>4,747,291</td>
</tr>
<tr>
<td>Kyber.CPA.Imp.Dec</td>
<td>1,420,367</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Gen \ (HW-AES: PRF/XOF; SW-SHA3: H)</td>
<td>14,512,691</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Enc \ (HW-AES: PRF/XOF; SW-SHA3: G, H)</td>
<td>18,051,747</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Dec \ (HW-AES: PRF/XOF; SW-SHA3: G, H)</td>
<td>19,702,139</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Gen \ (HW-AES: PRF/XOF; HW-SHA-256: H)</td>
<td>3,980,517</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Enc \ (HW-AES: PRF/XOF; HW-SHA-256: G, H)</td>
<td>5,117,996</td>
</tr>
<tr>
<td>Kyber.CCA.Imp.Dec \ (HW-AES: PRF/XOF; HW-SHA-256: G, H)</td>
<td>6,632,704</td>
</tr>
</tbody>
</table>

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


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


Saber.


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


Mike Hamburg.

Three bears.


David Harvey.

Faster polynomial multiplication via multipoint kronecker substitution.


Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware.

T. Pöppelmann and T. Güneysu.
Area optimization of lightweight lattice-based encryption on reconfigurable hardware.

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

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

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.

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

Joachim Von Zur Gathen and Jürgen Gerhard.
Modern computer algebra.